DOCUMENT RESUME

ED 098 963 IR 001 342

AUTHOR Brown, John Seely; And Others
TITLE Sophisticated Instructional Environment for Teaching Electronic Troubleshooting. Final Report.
INSTITUTION Air Force Human Resources Lab., Brooks AFB, Texas.
REPORT NO AFHRL-TR-74-77
PUB DATE Oct 74
NOTE 146p.; Report period covered January 1973-March 1974

EDRS PRICE MF-$0.75 HC-$6.60 PLUS POSTAGE
DESCRIPTORS *Computer Assisted Instruction; Computer Oriented Programs; *Electronic Equipment; Electronic Technicians; Program Descriptions; *Programing; *Simulation; *Technical Education
IDENTIFIERS *Air Force; Natural Language Programing; SOPHIE; Sophisticated Instructional Environment

ABSTRACT

A programing approach was used to implement a simulated laboratory training situation in which a student is allowed to troubleshoot a defective regulated power supply. The ways in which students can use natural language to ask questions about and manipulate the simulated device are described. The techniques developed to recognize English, to simulate the electronic circuit, and to model the student's knowledge about the circuit are explained. A conclusions section explains the generality of the work performed and possible extensions of the techniques to other training situations. (Author)

AFHRL-TR-74-77

AIR FORCE HUMAN RESOURCES LABORATORY

SOPHISTICATED INSTRUCTIONAL ENVIRONMENT FOR TEACHING ELECTRONIC TROUBLESHOOTING

By

John Seely Brown
Alan G. Bell
Richard R. Burton
University of California, Irvine
Irvine, California 92664

TECHNICAL TRAINING DIVISION
Lowry Air Force Base, Colorado 80230

October 1974
Final Report for Period January 1973 – March 1974

AIR FORCE SYSTEMS COMMAND
BROOKS AIR FORCE BASE, TEXAS 78235

NOTICE

When US Government drawings, specifications, or other data are used for any purpose other than a definitely related Government procurement operation, the Government thereby incurs no responsibility nor any obligation whatsoever, and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This final report was submitted by the University of California, Irvine; Irvine, California 92664, under contract F41609-73-C-0006, project 1121, with Technical Training Division, Air Force Human Resources Laboratory (AFSC), Lowry Air Force Base, Colorado 80230. Mr. Edward M. Gardner, Computer Based Systems Branch, was the contract monitor.

This report has been reviewed and cleared for open publication and/or public release by the appropriate Office of Information (OI) in accordance with AFR 190-17 and DoDD 5230.9. There is no objection to unlimited distribution of this report to the public at large, or by DDC to the National Technical Information Service (NTIS).

This technical report has been reviewed and is approved

MARTY R. ROCKWAY, Technical Director
Technical Training Division

Approved for publication.

HAROLD E. FISCHER, Colonel, USAF
Commander

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFHRL-TR-74-77	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) SOPHISTICATED INSTRUCTIONAL ENVIRONMENT FOR TEACHING ELECTRONIC TROUBLESHOOTING		5. TYPE OF REPORT & PERIOD COVERED Final January 1973 – March 1974 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) John Seely Brown Alan G. Bell Richard R. Burton		8. CONTRACT OR GRANT NUMBER(s) F41609-73-C-0006
9. PERFORMING ORGANIZATION NAME AND ADDRESS University of California, Irvine Irvine, California 92664		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62703F 11210205
11. CONTROLLING OFFICE NAME AND ADDRESS Hq Air Force Human Resources Laboratory (AFSC) Brooks Air Force Base, Texas 78235		12. REPORT DATE October 1974 13. NUMBER OF PAGES 144
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Technical Training Division Air Force Human Resources Laboratory Lowry Air Force Base, Colorado 80230		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

This research was completed under sub-contract with Bolt Beranek and Newman, Inc., Cambridge, MA.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

simulation
electronic equipment
technical training
Computer Assisted Instruction
computer programming
electronic troubleshooting
electronics training
SOPHIE
SOPHisticated Instructional Environment

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes the programming approach used to implement a simulated laboratory training situation in which a student is allowed to troubleshoot a defective regulated power supply. The ways in which students can use English to ask questions about and manipulate the simulated device are described. The techniques developed to recognize English, to simulate the electronic circuit, and to model the student's knowledge about the circuit are explained. A conclusions section explains the generality of the work performed, and possible extensions of the techniques to other training situations.

SUMMARY

Problem

The use of fonted electronics equipment to teach students how to isolate defects and repair them has caused numerous problems in the classroom. An instructor must insert faults manually and monitor students to insure that they do not further damage equipment while they are trying to fix it. Electronic devices are not built for training and are often delicate, and of course do not 'teach' the student other than to provide experience. In addition, it is not cost effective to provide adequate instructors to monitor each student and to tell him whether he is using proper trouble-shooting techniques as he operates.

Approach

Sophie (Sophisticated Instructional Environment) was developed to simulate electronics equipment with a computer program; it thus cannot be damaged by student error except as intended for realism. Sophie was designed to recognize major classes of English discourse about the instrument so that students could 'talk' about the instrument without learning special notations or encodings; Sophie further produces all responses in English for the same reason. Sophie was given an understanding of the circuit so that the student can find out whether his proposed fixes are reasonable on the basis of his measurements. A student is thus allowed to talk with Sophie about the circuit, change components, take measurements, and request evaluation of his performance in finding the inserted fault. The program has been tested by informal users and is awaiting test, using actual students from an Air Force electronics training course.

Findings

The highly complex program was found to typically consume less than 3% of the processor capacity of a medium size computer while running interactively, and to produce response times within about 2 seconds from requests; this represents greater than an order of magnitude over previous efforts in this area. English acceptability was very good and English output was terse but completely readable. The program has been observed to be significantly better than human observers in isolating faults randomly inserted and much more astute than students at finding the flaws in their trouble-shooting reasoning. The techniques developed for recognizing and processing English are very good and have already been applied to two other Air Force projects. The generality of the technique to other subject areas should be investigated further, particularly the use of capabilities such as Sophie as an author aid for producing other less sophisticated but more cost-effective types of instructional material.

TABLE OF CONTENTS

Chapter 1

Introduction

Objectives and Background

Although digital computers have become increasingly more powerful and versatile, their use in instruction has grown primarily in but one dimension -- that of finding cost-effective ways of providing more students with access to frame-oriented CAI systems. This report describes a research project pursuing a different dimension which has culminated in a system named Sophie.* Sophie takes full advantage of the symbol manipulation capabilities of advanced computer systems and advances the frontier of providing a qualitatively new kind of instructional environment.

The basic challenge in creating Sophie is that of endowing a CAI system with sufficient symbolic knowledge, problem solving strategies, and natural language capabilities so that it can mimic many of the capabilities of a human tutor. For example, we want Sophie to be able to respond, on its own, to a student's questions, evaluate his hypotheses, and provide detailed feedback about his answers. In short, we are trying to utilize many of the concepts and techniques of Artificial Intelligence in building a truely intelligent CAI system that exhibits a sense of

---

*A Sophisticated Instructional Environment

"understanding" about the subject domain it is teaching.

Over the last few years there have been several research groups building CAI systems that have some of the above-mentioned capabilities. GEO-SCHOLAR (Car73) is probably the best known of these systems and was the first to demonstrate the feasibility of this kind of instructional system. However, GEO-SCHOLAR had some serious limitations. For example, its parser was primarily keyword driven and therefore could not begin to handle complex statements or questions posed by the student. Also, its problem solving abilities were confined to techniques which operated on semantic networks, a data type which captures some kinds of information exceedingly well but other kinds of information such as procedural not as well.*

Sophie, unlike GEO-SCHOLAR, uses several representations of knowledge including semantic/conceptual networks and has inferencing procedures which are specially designed for each of these representations. Much of Sophie's power arises out of the interaction of these representations although the main seat of intelligence in Sophie resides in its ability to draw conclusions and make inferences from setting up, "running", and examining (abstracting) a simulation model of its problem domain. We

---

*Recent research on SCHOLAR is alleviating some of these limitations.

believe that Sophie obtains so much leverage from inferencing based on "intelligently" invoked simulation that it simultaneously satisfies two virtually contradictory goals. On one hand, it can produce deep inferences which enable it to answer questions which even human tutors would find extremely difficult to answer. On the other hand, it is sufficiently fast and complete that it can be thoroughly exercised in an instructional environment.

In order to illustrate the philosophy behind Sophie the general domain of electronic knowledge was chosen and the initial task of building a system for teaching electronic troubleshooting was started.

Reasons for Choosing Electronic Troubleshooting

There are basically three reasons influencing our choice of electronic troubleshooting as the subject domain around which to build this system.

The first concerns the belief that troubleshooting or debugging reflects a qualitative, common sense kind of reasoning that has never been satisfactorily studied. This kind of reasoning does not have its roots in axiomatic logics or exact quantitative reasoning. A good repairman draws useful conclusions without even knowing the logical assumptions underlying much of his reasoning. The rational basis for his cognitive skill reflects much more the

property of abduction in that the important question is how to merge a few examples (or measurements) with an underlying logic to arrive at a set of viable sub-theories or hypotheses. This kind of process underlies not only troubleshooting but a great deal of human intelligence. Hopefully, Sophie will provide us with a tool and a controlled environment to experiment more fully with this kind of reasoning.

The second reason that this domain was chosen is that the typical laboratory for learning electronic trouble shooting has several severe limitations. Such a laboratory usually consists of several pieces of working equipment into which the instructor can insert faults. However, the class of faults that he can insert is artificially limited by practical considerations* and in some cases, he can fail to predict the full ramifications of his inserted fault leading to other components being damaged. Likewise, when the student replaces a component, either it or other components are apt to be destroyed in the operation, the former happening when he has failed to locate the fundamental fault. Another kind of limitation is that a great deal of

---

*Examples of such considerations might be: can the fault be easily triggered off by simply setting a switch, can it be easily detected by carefully examining the circuit to see what solder joints have been tampered with, can the faulted component be easily obtained if it is to replace a working component, etc.

the student's time is ineffectually used by his having to make time consuming measurements which, for example, might require cutting wires and removing elements from the circuit, etc. Finally, the instructor seldom has the time to have the student articulate his hypotheses as he goes about troubleshooting and then to evaluate those hypotheses or to answer questions that arise in the student's mind as he is actually working with the instrument. A computer based problem solving "laboratory" could circumvent all these limitations.

The third reason relates to our being able to design especially powerful inference procedures to perform the kinds of deductions required for handling questions that arise in this domain. We shall explore this issue in depth later in the report.

Scenario

Our system is based around the scenario of a student attempting to isolate a fault in a given instrument. In this setting, Sophie would present the student with a circuit schematic of an instrument and, if requested, would automatically select and insert a fault of some specified degree of difficulty. The student would then try to isolate the fault by requesting various measurements under any instrument setting that he desires. At any time he can

offer an hypothesis about what he thinks is wrong with the instrument and have the system evaluate his hypothesis. This evaluation would report to the student whether his hypothesis is consistent with what he should have learned from his measurements. (Of course, the particular set of measurements is not known ahead of time.) The student could also, at any time, replace a given component, but before it is replaced he would be queried as to what he thought was wrong with it. If his answers were correct, the component would be replaced. In those cases where he had only discovered a fault caused by a deeper fault, the replaced component would be reblown until he discovered the fundamental fault. If the student becomes stuck and cannot think of any faults which would explain the measurements he has made, he can ask for help. Sophie would then generate possible hypotheses which the student could explore.

Goals

In order to engender this kind of scenario, Sophie has to understand enough about a given instrument to be able to derive the answers to any measurements the student might request. Likewise it has to accept any hypothetical fault about any component in the instrument and derive the results of requested measurements in the context of that faulted circuit. In addition to measurements, it has to infer answers to questions concerning properties of a given

component and determine if and when a given component might blow because of another component blowing. Most importantly, it has to be able to intelligently evaluate a student's hypothesis about what might be wrong. Simply checking if the hypothesis is in fact true (i.e. is it the actual fault) would not suffice since there may be several theories about possible faults that are consistent with the set of measurements he has thus far taken. In addition, even when an hypothesis is inconsistent with his observations, it must inform him as to the nature of this inconsistency. Finally, Sophie should appear "friendly" by allowing the student to communicate with it in a natural subset of English and should respond fairly quickly to his requests.

Performance

Sophie has met all of these goals and has exceeded our expectations in terms of its speed. The average amount of CPU time it uses to parse a student's question, semantically interpret this parse, and derive an answer to his question or measurement takes, on the average, less than two seconds on a PDP-10 running TENEX (Bob72). Hypothesis evaluation takes several seconds longer depending on how many measurements the student has currently made and how many of these measurements need to be considered for refuting or supporting his hypothesis.

Implementation

Sophie has been implemented primarily in INTERLISP (Tei74) but includes a general purpose circuit simulation system written in Fortran. The LISP portion takes almost the full 256k address space (of which slightly more than half is the LISP system itself). The simulation system takes an additional 40k and runs in a separate address under the control of the LISP portion. The simulator currently contains a model of the Heathkit IP-28 regulated power supply which is a reasonably sophisticated, six transistor, current and voltage limiting power supply. (See Appendix 1 for a description of the IP-28.)

Overview of Sophie

Upon entry into Sophie, the student receives a prologue describing the instrument being modelled and giving him a brief summary of the system's deductive and linguistic capabilities. The student then can ask the system to randomly generate an easy or hard fault or he can perform experiments in the working circuit in order to better understand the internal operation of the instrument. He can also request that any component of the circuit be modified or faulted in any reasonable way and on this modified circuit he can either perform experiments or examine how a fault or modification may have caused other components in

the circuit to blow. In addition to active measurements, he can request factual information about any part in the instrument such as the beta of a given transistor or the power rating of a resistor, etc.

If he has chosen to have Sophie insert a fault, he can then attempt to isolate the fault or faults by requesting various experiments be run on the faulted instrument (i.e. by making measurements). At any time he can ask that the measurement he has just made in the faulted circuit be repeated on a correctly functioning circuit in order to determine if his measurement reflects a symptom, and if it does, to find out how great the discrepancy is between the two values.

While the student is using Sophie to troubleshoot an unknown fault, he can request that a part be replaced, ask if a part is bad (or if anything is still bad), propose for evaluation an hypothesis about what is wrong, or request help if he is at a loss in conjuring up some hypotheses which are consistent with all his measurements. If in the midst of a troubleshooting session he has a deep question or an idea about the interaction of components in the circuit he can SAVE the context of his current scenario and invoke a fresh version of Sophie on which to perform an experiment. Later, he can return to his old context (RESTORE) and continue his troubleshooting session. The hypothesis

evaluator and reviewing features of Sophie are correctly keyed into this SAVE-RESTORE feature so that evaluation is only performed on those measurements performed while actually trying to troubleshoot the given fault. Finally, the student can ask Sophie questions about what happens if a particular component is bad, etc.

In order to illustrate some of Sophie's capabilities, we include below an annotated protocol generated by a student using Sophie in the troubleshooting mode.

PROTOCOL

** Parts typed by the student are underlined.
**comments are in script type.

WOULD YOU LIKE TO SEE THE PROLOGUE?
> YES

WELCOME TO SOPHIE, A SIMULATED ELECTRONICS LABORATORY. THE PIECE OF EQUIPMENT YOU ARE STUDYING IS THE IP-28 REGULATED POWER SUPPLY. WOULD YOU LIKE ME TO PRINT A CIRCUIT SCHEMATIC FOR YOU?
> NO

SOPHIE HAS A CORRECTLY FUNCTIONING IP-28 WHICH YOU ARE FREE TO EXPLORE. YOU CAN TAKE MEASUREMENTS IN THE WORKING CIRCUIT; FAULT THE CIRCUIT (E.G. "SUPPOSE THE EMITTER OF Q6 OPENS") AND THEN TAKE MEASUREMENTS TO SEE HOW THE FAULTED CIRCUIT BEHAVES; OR ASK ME TO "INSERT A FAULT" AND TRY TO DISCOVER WHAT IS WRONG. IF YOU THINK YOU KNOW WHAT IS WRONG, YOU CAN "REPLACE" THAT PART AND SEE IF YOU WERE CORRECT. WHILE YOU ARE TROUBLESHOOTING, THE LABORATORY HAS A TUTOR ON-CALL WHO CAN COMMENT ON THE CONSISTENCY OF A PARTICULAR HYPOTHESIS WRT THE MEASUREMENTS YOU HAVE MADE. YOU CAN INVOKE THE TUTOR BY PROPOSING YOUR FAULT (E.G. "COULD THE PROBLEM BE THAT R22 IS OPEN").

```
WOULD YOU LIKE TO START WITH A FAULTED INSTRUMENT WHICH NEEDS FIXING?
> YES
```

System randomly selects a fault, modifies the instrument, and is now ready to answer questions about the malfunction.

```
THE INSTRUMENT HAS A FAULT
AND ITS CURRENT SETTINGS ARE:
CC        1.0       (CURRENT CONTROL, FROM 0.0 (MIN CURRENT) TO 1.0)
CR        HIGH      (CURRENT RANGE SWITCH, HIGH=1 AMP, LOW=.1 AMP)
LOAD      1000.0    (LOAD RESISTANCE IN OHMS)
VC        1.0       (VOLTAGE CONTROL, FROM 0.0 (MIN VOLTAGE) TO 1.0)
VR        HIGH      (VOLTAGE RANGE SWITCH, HIGH=30 VOLTS, LOW=10 VOLTS)
NOW YOU CAN REQUEST MEASUREMENTS TO LOCATE THE FAULT.

>>WHAT IS THE OUTPUT VOLAEG
=VOLTAGE
```

Example of spelling correction -- especially useful if he is using a 33 teletype or there is communication noise.

```
THE OUTPUT VOLTAGE IS 30.03 VOLTS.
```

This is the output voltage across a 1000 ohm load when the malfunctioning instrument is set as above.

```
IF YOU ARE NOT SURE WHETHER A VALUE IS CORRECT, YOU CAN ASK 'IS THAT
VALUE CORRECT?'
```

Student wants to know if this measurement reflects a symptom.

```
>>IS THAT CORRECT?
IN A WORKING CIRCUIT THE OUTPUT VOLTAGE IS 30.07 VOLTS.
THAT VALUE IS CORRECT.

>>CHANGE THE LOAD RESISTANCE TO 30 OHMS
THE LOAD IS 30.0
```

Put on a new load resistance.

```
>>NOW WHAT IS THE OP VOLTAGE
THE OUTPUT VOLTAGE IS 1.82 VOLTS.
```

"OP" is an abbreviation for output.

```
>>WHAT SHOULD IT BE?
IN A WORKING CIRCUIT THE OUTPUT VOLTAGE IS 29.88 VOLTS.
```

Student has now detected a symptom!

```
>>WHAT IS THE VBE OF THE CURRENT LIMINTG TRANSISTRO
=TRANSISTOR
=LIMITING
THE BASE EMITTER VOLTAGE OF Q6 IS 36.23
 MILLIVOLTS.
```

→ *(Q6 should be off)*

```
>>WHAT IS THE COLL CURRENT OF THE VOLTAGE LIMITING TRANSISTOR
THE COLLECTOR CURRENT OF Q5 IS 0.0 AMPS.
```

→ *(Q5 should be off)*

```
>>WHAT IS THE V BETWEEN NODE 4 AND NODE 5
THE VOLTAGE BETWEEN N4 AND N5 IS .77 VOLTS.

>>COULD THE BETA OF Q4 BE TOO LOW
I NEED A PARTICULAR VALUE FOR THE BETA OF Q4
IS IT OKAY IF I USE 20 ?
> OK

HMMM, LET ME THINK ABOUT THAT FOR A MINUTE.

WHEN THE SETTINGS WERE:
   CC=1.0, CR=HIGH, LOAD=30.0, VC=1.0, VR=HIGH

THAT FAULT CONFLICTS WITH THE FOLLOWING
 MEASUREMENT,
YOU OBSERVED THAT THE VOLTAGE BETWEEN
 N4 AND N5 WAS .77 VOLTS.
IF THE BETA OF Q4 WERE 20 IT WOULD BE 1.54 VOLTS,
IN A WORKING CIRCUIT IT IS 1.7 VOLTS.

ALSO:
THAT FAULT CONFLICTS WITH THE FOLLOWING
 MEASUREMENT,
YOU OBSERVED THAT THE OUTPUT VOLTAGE WAS 1.82
 VOLTS.
IF THE BETA OF Q4 WERE 20 IT WOULD BE 6.94 VOLTS,
IN A WORKING CIRCUIT IT IS 29.88 VOLTS.
BUT THAT FAULT WOULD ACCOUNT FOR SOME OF
THE MEASUREMENTS UNDER THOSE SETTINGS.

SHALL I CONTINUE COMMENTING ON YOUR HYPOTHESIS?
> NO THANKS


>>SAVE
CONTEXT SAVED, TO RETURN TO IT TYPE 'RESTORE'.


>>SHORT R11
NOW R11 IS SHORTED

>>WHAT IS THE OUTPUT V
CONGRATULATIONS, Q2 JUST BLEW
CONGRATULATIONS, Q5 JUST BLEW
CONGRATULATIONS, D1 JUST BLEW
CONGRATULATIONS, D2 JUST BLEW
THE OUTPUT VOLTAGE IS .22 MILLIVOLTS.
```

Student generates an hypothesis.

System needs a specific value and then suggests one. Student can override suggestion and put in his own value.

Defines the context of his measurement. (In this case there were only two to choose from.)

This is the most important descrepancy and should call attention that probably one of two transistors is totally malfunctioning.

This is a minor descrepancy although the actual voltage difference is much larger than above!

The student wishes to explore another hypothesis before continuing.

Making a hypothetical modification to the circuit.

The effect of the above modification causes Q2, Q5, D1 and D2 to blow.


```
>>WHAT IS WRONG
 D1 IS FAULTED OPEN
 Q2 IS FAULTED OPEN
 Q2 IS FAULTED EB/EC/BC = SH/SH/SH
 Q5 IS FAULTED EB/EC/BC = SH/SH/SH
 R11 IS FAULTED SHORTED
```

Tells the student the precise way the parts blew.

```
>>RESTORE
PREVIOUS SITUATION RESTORED.
```

The student has finished exploring his hypothesis and goes back to trouble-shooting.

```
>>REVIEW
        CC=1.0, CR=HIGH, LOAD=1000.0, VC=1.0, VR=HIGH,
THE OUTPUT VOLTAGE IS 30.03 VOLTS.
        CC=1.0, CR=HIGH, LOAD=30.0, VC=1.0, VR=HIGH,
THE OUTPUT VOLTAGE IS 1.82 VOLTS.
THE BASE EMITTER VOLTAGE OF Q6 IS 36.23 MILLIVOLTS.
THE COLLECTOR CURRENT OF Q5 IS 0.0 AMPS.
THE VOLTAGE BETWEEN N4 AND N5 IS .77 VOLTS.
```

Prints the results of his measurements in a condensed format. This information is stored on a history list.

```
>>WHAT ARE THE SPECIFICATIONS OF Q4
THE SPECIFICATIONS OF Q4 ARE;
BETA -- 100
POWER/RATING -- 6 WATTS

>>WHAT IS THE BASE CURRENT OF Q4
THE BASE CURRENT OF Q4 IS .61 MILLIAMPS.
```

Indicates that the beta of Q4 is about what it should be.

```
>>WHAT IS THE COLL CURRENT OF Q4
THE COLLECTOR CURRENT OF Q4 IS 60.58 MILLIAMPS.

>>WHAT IS THE BASE EMITTER V OF Q3
THE BASE EMITTER VOLTAGE OF Q3 IS .61 MILLIVOLTS.
```

Something is clearly amiss here. This voltage reading is essentially zero.

```
>>REPLACE Q3
I AM GOING TO ASK YOU SOME QUESTIONS ABOUT HOW Q3 IS FAULTED.
IF AT ANY TIME YOU WOULD LIKE TO MAKE MORE
 MEASUREMENTS, SIMPLY TYPE CONTROL-Z.

ARE ANY JUNCTIONS SHORTED?
> YES
WHICH JUNCTIONS ARE SHORTED?
>  THE BASE EMITTER JUNCTION
ANY OTHERS?
> NO
Q3 IS NOT FAULTED IN THAT MANNER.
Q3 NOT REPLACED.

>>IS IT POSSIBLE THAT R22 IS SHORTED
THAT FAULT IS CONSISTENT WITH ALL OF THE
 MEASUREMENTS YOU HAVE MADE.
```

This command is useful if the student realizes that he hasn't deduced the way a part had faulted.

Even if he answers this incorrectly but has identified any of the shorted junctions it would accept this and tell him the complete fault mode.

This doesn't mean that he has necessarily found the fault.

```
>>WHAT IS THE RESISTANCE OF R22
THE MEASURED RESISTANCE OF R22 IS 0.0 OHMS.
```

Checking to see if it is, in fact, the fault.

```
>>REPLACE R22
R22 REPLACED
```

This time Replace doesn't ask any questions since it figures he must know the exact way R22 is faulted.

021

```
>>NOW WHAT IS THE OUTPUT VOLTAGE
THE OUTPUT VOLTAGE IS 29.88 VOLTS.
```

Checks the output voltage.

```
>>IS THAT CORRECT
IN A WORKING CIRCUIT THE OUTPUT VOLTAGE IS 29.88 VOLTS.
THAT VALUE IS CORRECT.
```

```
>>IS ANYTHING ELSE WRONG
NO
```

Just checking to see if anything else is still wrong that possibly the last measurement wouldn't have detected.

```
>>RESTART
COMMENCING RESTART.
```

Closes the protocol file for this session. Initializes a new file.

```
TOTAL REAL TIME OF SESSION = 40 MINUTES
TOTAL CPUTIME OF SESSION = 58 SECONDS

WOULD YOU LIKE TO SEE THE PROLOGUE?
> NO

WOULD YOU LIKE TO START WITH A FAULTED INSTRUMENT WHICH NEEDS FIXING?
> NO

>>WHAT ARE THE SETTINGS OF THE CONTROLS
THE CURRENT SETTINGS ARE:
CC          1.0
CR          HIGH
LOAD        1000.0
VC          1.0
VR          HIGH
```

Student poses a hypothetical question.

```
>>IF THE BASE OF Q5 OPENS WHAT HAPPENS TO THE OUTPUT VOLTAGE
IS THE COMPLETE FAULT MODE EB/EC/BC = OP/OK/OP
 WHAT YOU INTENDED?
> YES
IF Q5 HAD FAULT MODE EB/EC/BC = OP/OK/OP
UNDER THE PRESENT SETTINGS, THE OUTPUT VOLTAGE IS 46.81 VOLTS.
IN A WORKING CIRCUIT IT IS 30.07 VOLTS.

>>WHAT HAPPENS TO THE OUTPUT VOLTAGE WHEN R9 OPENS
IF R9 WERE OPEN
UNDER THE PRESENT SETTINGS THERE IS NO CHANGE, I.E. 30.04 VOLTS.

LET ME TRY SOME OTHER SETTINGS.
UNDER THE SETTINGS:
        CC=1.0, CR=HIGH, LOAD=10.0, VC=1.0, VR=LOW
THE OUTPUT VOLTAGE IS 6.76 VOLTS.
IN A WORKING CIRCUIT IT IS 9.89 VOLTS.
```

Sophie searches for a new setting of the controls and load resistance to determine if R9 opening can affect the output voltage

```
>>STOP.
```

Tells the system he is ready to quit.

Examples of Sophie's Linguistic Capabilities

We have mentioned that Sophie must permit the student to request measurements and state hypotheses in a reasonable subset of English. Although we stress that its current linguistic capabilities are highly tuned to the subject domain and are far from handling most of the English language, we provide below some examples of student's statements that Sophie presently understands.

Requesting measurements: (parse times including semantic interpretation are placed in parentheses)

What is the voltage across the base emitter junction of the current limiting transistor? (140 ms)
What is the VBE of Q6? (120 ms)
What is current thru the base of Q5? (130 ms)
What is the IB of Q5? (100 ms)
What is the output voltage? (80 ms)
What is the voltage between node 1 and the positive terminal of C6? (280 ms)
What is the dynamic resistance of R11? (120 ms)
What is the power rating of R8? (100 ms)
What is the beta of the voltage limiting transistor? (110 ms)
What are the specs of Q3? (90 ms)
In a working circuit what is the output voltage of the power reference transformer? (90 ms)

Modifying the instrument:

Change the output load to 10 megaohms
Suppose the beta of Q5 is 200
Suppose the breakdown voltage of D5 is 30 volts
Let C2 be leaky
Turn up the voltage control
Set the voltage range switch to 30 volts
Set the current control to maximum
Suppose the BE junction of Q6 is shorted

Questions:

Is the current limiting transistor bad
What happens to the VBE of Q4 when R22 shorts
If R8 opens then what happens to the output voltage
What are the specifications of Q3
What is wrong
What happens if Q3 shorts

Hypothesis checking:

Is it possible that the breakdown voltage of D5 is too low
Could the problem be that C2 is leaky
Could the beta of Q4 be too low
Replace R6

Noun phrase utterances: (noun phrases get interpreted as questions)

Voltage between the base of Q5 and the wiper of R7
Output voltage
VBE of Q6
I thru C6

Miscellaneous commands:

Review
Remove all faults
Reset the instrument
Restart
Save
Restore
Stop
Review all

Possible Uses of Sophie

Although Sophie may be viewed as a CAI system in its own right, we prefer to view it as a set of powerful tools with which to implement various teaching strategies. For example, one can imagine this system being used in conjunction with a frame-oriented system which would be

responsible for presenting textual material about the operational principles of a given instrument. Once this material was mastered, problems would be presented to the student for which the unique power of Sophie would best be suited. These problems could involve faulting the instrument in ways which demonstrate the interactions of the component which the student has just been told about, or they could simply allow him to fully explore the internal states of a given section of the instrument.

A more exciting possibility involves a gaming situation. After students are exposed to the fundamentals of how a given circuit operates, they would participate in a two-person game wherein one student introduces a subtle fault into the circuit and predicts the consequences of his modification. The other student must then discover the modification by performing a series of measurements. Each measurement has a cost* and the total cost is computed for his sequence of measurements. After the fault is isolated the roles are reversed and the game is played again.

Another version of this game is also possible. After a

---

*The cost could be varied to encourage students to learn different methods of troubleshooting but would usually reflect the difficulty involved in actually making the measurement in a real electronics laboratory, i.e. external measurements are the cheapest while ones which would require removing a component from the circuit are expensive.

modification is introduced by one student, the other proposes measurements which the first student answers as best he can on the basis of his earlier predictions of what the circuit will do given the modification he introduced. The system scores both students: one on the basis of relevancy, cost, and type of the measurement he asks for, and the other in terms of accuracy of the predicted answer. Many variations of this game are possible. For example, both scores might be made visible to both players (or only their own score, or neither), or correct answers might be provided by the system to both players (privately or publicly). Moreover, the system could exercise some judgment and interrupt the answerer if a mistake by him might result in a potentially disastrous compounding of misunderstandings.

Although the gaming scenario may seem of primary relevance for diagnostic training, it has a far greater importance in providing the student with an intuitive understanding of the qualitative and causal behavior of system components, etc. In fact, one of the best ways of discovering the purpose for a particular component is to "alter" that component and see what aspects of the circuit's behavior changes.

Chapter 2

Natural Language Understanding

Introduction

As instructional systems grow in their ability to answer questions and evaluate hypotheses, the need for parsing techniques more powerful than ad hoc keyword analysis schemes becomes a necessity. This is not to say that there is no place for keyword analyses in CAI for they can be very useful in highly predictable environments. But there are major limitations to keyword parsers and in many cases they actually become more cumbersome and inefficient than a well designed structural parser. For example, although a keyword analysis might suffice in decoding the utterance:

"What is the output voltage?"

such a technique begins to get messy if "output voltage" can have numerous modifiers such as:

"What is the output voltage of the power reference transformer?"

If these modifiers can in themselves be modified, the situation quickly grows out of hand for even the most advanced ad hoc keyword system. As an example of such a

situation consider the question:

"What is the voltage between the anode of D6 and the collector of the voltage limiting transistor?"

Our approach to constructing a front-end natural language processor was heavily influenced by pragmatic considerations. We needed a parser that would be practical to use in a CAI environment and one which would take only a few man-months to complete. Our first choice was to use one of the most powerful and clean parsing systems currently available -- namely Woods' LSNLIS parser. (See (Woo72a)(Woo72b) for descriptions of this parser.) Although our slightly trimmed down version of this system proved to be surprisingly efficient, we eventually rejected using it because it produced a structural description of a question which was more detailed than our semantic routines could take advantage of. Although we could have further simplified Woods' grammar, etc. we decided to explore a fundamentally different approach.

A Semantically Driven Fuzzy Parser

After studying numerous protocols of students using a mocked-up version of Sophie, we noticed the powerful constraints that existed in the relationships between the various semantic/conceptual entities making up a question.

For example, if one asks for a voltage measurement it is either between two nodes, across a particular component, or across some output terminals. It seemed that this high degree of semantic predictability could be utilized by a predictive analyzer ("parser") by simply refining the usual syntactic categories such as noun phrase into relevant semantic/conceptual categories such as "measurement". In general, such refinements could lead to a phenomenal proliferation of categories to be captured by the "grammar", but an analysis of our data indicated that such an approach was feasible. These and other considerations lead us to build a highly efficient top-down (goal-oriented), context free parser which makes its predictions on the basis of semantic rather than syntactic categories.

Our natural language processor incorporates a certain dimension of "fuzziness". If at a given moment in a parse it is searching for a particular instantiation of a semantic category, and the word currently being pointed to fails tc atisfy this instantiation, it skips over that word and continues searching.* This means that if the student uses certain words or concepts that the system doesn't know about, it can ignore these words and try to make sense out of what remains. However, this kind of "fuzziness" is not

---

* The number of words that can be skipped over is controlled by the particular semantic category being searched for.

always a blessing. Since words can be skipped over during the parsing, there is always the possiblity of mis-parsing a sentence. This problem requires either placing a severe constraint on the ordering of grammar rules, or it requires finding all possible parses of a sentence in order to discover the parse that accounts for the greatest number of words in the sentence. Because the second of these two alternatives is potentially very time consuming, we use the first technique. To limit the negative consequences which may result from a misunderstood question, Sophie responds to a student's question with a full sentence which tells the student what question is being answered.

Limitations

By restricting ourselves to a context free type of parser, we obviously sacrificed capturing (in any reasonable way) transformational paraphrases of an utterance. Instead, we settled for allowing only one or two ways -- hopefully the simplest ways -- of saying or requesting anything that the question answering component of our system could handle.* For example, our system parses the question:

---

*Certain paraphrase capabilities were explicitly put into the natural language processor. For example, one can either ask for the voltage across the base emitter of a transistor or for the base emitter voltage of a transistor.

What is the beta of the current limiting transistor?

but does not parse the same question paraphrased as:

What is the beta of the transistor which limits the current flow?

We realized that by not accepting many of the syntactic paraphrases of an utterance, the naturalness of our system could be drastically reduced. However, we hoped that the system would accept enough "English" that the natural flow of communication would not be impaired thus enabling the user to get involved with solving a problem rather than with searching for ways of saying or asking for something.

"Naturalness" of the Language Capabilities

In order to study how easily users could adapt to these linguistic limitations, we collected well over a hundred hours of protocols of people using Sophie from various ARPA sites. Each user had seen a protocol of a "typical" session which gave him some idea of the system's linguistic and logical capabilities. Initially, anytime Sophie encountered a sentence which it could not parse, that sentence was automatically stored on a file which was later used to provide data for expanding our grammar. A point has now been reached in which Sophie handles nearly all sentences generated by users who have had at least one prior session

with the system.

These experiences convinced us that for our highly constrained domain our approach to parsing was viable. Although extensively handling paraphrases would surely have helped our system appear more natural, three other issues seemed at least as important. The first is the need for handling abbreviations such as VBE standing for "base emitter voltage". Such a facility has now been added to our natural language processor. The second is the need for a spelling corrector, a facility which can greatly reduce the amount of concentration, and hence effort, that a poor typist expends in addressing the system. Our natural language processing system now utilizes the spelling correction algorithms provided by the INTERLISP DWIM facility (Tei74). The third is much more problematic. It concerns the issue of handling context dependent anaphoric references and ellipses, e.g. pronoun references referring to a prior sentence. Having this capability appears to be especially crucial when the user has become totally immersed in using the system as opposed to simply trying it out. Some examples will illustrate how helpful such a facility could be:

Example 1:

What is the voltage across the base emitter junction of the current limiting tansistor?

What is the current through it?
("It" refers to "the base emitter junction of the current limiting transistor.")

Example 2:

What is the current through the base of Q6?

What is it through the emitter?
("It" refers to "current" and "Q6" is implied but not mentioned.)

What about through the collector?
(In this case, "current" and "Q6" are both implied but neither is mentioned.)

Example 3:

What is the output voltage?

What is the voltage control set to?

Is that correct? ("That" refers to "the output voltage" in the earlier statement.)

A solution to this problem is extremely complex and is beyond the scope of Sophie.

Description of the Parsing Process

The parsing of a student's response begins with a scanning of the complete string. During this scan, any abbreviations in the string are expanded into their full form. In addition, a mechanism for handling contractions and run-ons is provided to allow for the expansion of such words as "what's" and "whatis". Compound words are rewritten as single entities, if possible, because if left until parsing, they would complicate the grammar and might require backing up to disambiguate. For example, in the question, "what is the voltage range switch setting,"

"voltage range switch" is rewritten as "VR." If not rewritten, "voltage" would be mistaken as a noun and an attempt would have to be made to parse "range switch setting" as a place to measure voltage. Of course after this failed the correct parse would be found but reducing compound words avoids the otherwise necessary computation.

Another operation performed during the prescan is cursory spelling correction. Spelling correction is attempted on any word of the input string which the system does not recognize. The spelling correction algorithm that is used takes the (possibly) misspelled word and a list of correctly spelled words and determines which (if any) of the correct words is close to the misspelled word (using a metric determined by number of transpositions, doubled letters, dropped letters, etc.). During the prescan, the list of correct words is very small (approximately a dozen) and is limited to very commonly misspelled words and/or words which are critical to the understanding of a sentence. The list is kept small so that the time spent attempting spelling correction, prior to attempting a parse, is kept to a minimum. Remember that the parser has the ability to ignore words in the input string so we do not want to spend a lot of time correcting a word which won't be needed in understanding the statement. But notice that certain words can be critical to the correct understanding of a statement. For example, suppose that the phrase "the base emitter

current of Q3" was incorrectly typed as "the bse emitter current of Q3". If "bse" were not recognized as being "base" the parser would ignore it and (mis-)understand the phrase as "the emitter current of Q3".* Because of this problem, words like "base", are considered critical and their spelling is corrected before any parse is attempted. Note that there are a lot of words (e.g. "capacitor", "replace", "open", etc.) which if misspelled would prevent the parser from making sense of the statement but would not lead to any mis-understandings. These words are therefore not considered to be "critical" and would be corrected in the second attempt at spelling correction which is done after a statement fails to parse.

After a student's statement has been prescanned, an attempt is made to parse it using an embodiment of the grammar listed in Appendix 2. The top-level rule is given in Figure 2.1.

Figure 2.1

<STATEMENT> := <REQUEST> ! <SET> ! <MODIFY>

As mentioned earlier, our grammar uses non-terminals which represent semantic/conceptual categories. This rule

---

*To minimize the consequences of such mis-interpretation, the system always responds with an answer which indicates what question it is answering, rather than just giving the numeric answer.

indicates that any statement which the student makes to the system is either 1) a request for information of some kind, <REQUEST>, 2) a command to modify the circuit model in some way, <MODIFY>, or 3) a command to change the settings of one of the controls, <SET>. To determine if the statment is a request, Sophie calls the function <REQUEST>.* <REQUEST> looks at the input string and calls other functions, (for instance, <MEASUREMENT>) to decide whether or not the string is a request for information. If the input string is a request (that is, parses into a <REQUEST>), the function <REQUEST> returns a program (function call) which represents the "meaning" of the student's question. This program is then executed (EVALed) to answer the question. If the statement is not a request, Sophie calls the function <MODIFY> and then the function <SET> to determine if the statement parses to either a modification command or a control changing command. From either <MODIFY> or <SET> (whichever is successful), a function call is returned which when EVALuated, performs the desired commands.

If neither <REQUEST>, <SET>, nor <MODIFY> is successful, the statement does not parse and two contingency methods are tried. The first attempt made is to put a "What is" on the front of the input string and then to see if the

---

*This is because the grammar is written directly as LISP functions which incorporate the parser. This concept will be explained in more detail later.

resulting string is an acceptable <REQUEST>. (This abbreviated form of question-asking shortens the amount of typing the student must do.) If this fails, Sophie uses the spelling correction program that was used in the prescan, but this time with a much larger list of words (approximately 125). If any misspellings are discovered, the statement is re-processed.

The above discussion glosses over two very important points: 1) how the parse is actually performed and 2) how the semantic form representing the "meaning" is determined.

The Parser

Most parsing systems make a distinction between the grammar and the mechanisms for interpreting the grammar. This is done for economy of expression and clarity. It is important to choose a formalism in which to express the grammar which has enough freedom to allow the grammar writer to express his rules naturally and concisely, but provides enough structure that the mechanisms for handling the rules can still be done efficiently. Some examples of grammar formalisms are BNF, ATNs (Woo72a), PROGRAMMER (Win73) and pattern matching languages for transformational grammars. Our system expresses the grammar directly as LISP functions. This means that the language which the grammar accepts is determined by evaluating the grammar itself (i.e. the

grammar becomes the parser!). Since the grammar is written in LISP it provides a natural method of ordering the application of grammar rules and allows the grammar to be compiled which makes the resulting parser very efficient. The back-up required by the parser is handled by using the normal LISP control structure via COND, AND and OR functions.

When writing the grammar, the rule for each non-terminal is written as a single function which takes into account all possible ways of expressing that semantic category. Each of the grammar functions is a function of one argument, the string which the function (non-terminal) is to accept or reject. For example, the non-terminal <NODE> is represented as a LISP function <NODE> which can be called by any other LISP function to, in effect, answer the question "Does the string starting at this point parse into a <node>?"*

There are generally two types of checks that a rule-function performs. One is a check for the occurrence of a word or words which satisfies certain predicates in the input string. This checking is done with two functions -- CHECKLST and CHECKSTR. CHECKLST looks for a word in the string matching any of a list of words. CHECKSTR looks for

---

*This is similar to the concept of a push to another network in the ATN framework.

a word in the string satisfying an arbitrary predicate. It is through these functions that the parser gets its fuzziness. For example, if CHECKSTR is called with the string "resistor R9" and a predicate which determines if a word is the name of part (e.g. "R9"), CHECKSTR will succeed by skipping the word "resistor", which is, in this case, a noise word. The other usual type of operation performed by the grammar rules is to check for the occurrence of other non-terminals. This is done by calling the proper function (grammar rule) and passing it the correct position in the input string.*

If the grammar rule is successful (if the string parses into the correct semantic category), the rule-function passes back two pieces of information. First, it returns some indication of how much of the input string it accounted for, (that is, where it stopped). The convention adopted is that a grammar rule returns as its value a pointer to the last word in the string accepted by that rule. Second, the non-terminal passes back a structural description of the phrase that was just parsed. This structure is passed back in the free variable RESULT (analogous to Woods's "*" upon return from a PUSH).

---

*There are no restrictions placed on the grammar rules and a function can do whatever is necessary or efficient to recognize an occurrence of its semantic category.

Semantic Interpretation

The structural description of a phrase returned by a grammar rule is a piece of LISP code which when evaluated represents the meaning of that phrase. Each semantic category in the grammar has a corresponding set of functions or class of objects in the semantics of the system which is the reason for the existence of that semantic category. Each grammar rule knows enough about its semantics to construct the proper function call for any of its various phrasings. In other words, the semantic interpretation of the parse tree is occurring during the parsing.* For example, consider the non-terminal <MEASUREMENT> shown in Figure 2.2. The reason for this non-terminal is to parse all of the ways that a student can specify a measurement (voltage across D3, output current, etc.). To make a measurement the system needs a quantity to measure <MEAS/QUANT> (voltage, current, resistance, power dissipation), and something to measure with respect to (e.g. a part, <PART/SPEC>; a transistor junction, <JUNCTION>; or possibly a point in the circuit, <NODE>). So the rule for <measurement> expresses all of the ways that the student can give a measurable quantity plus supply its required arguments. The structure which results from <MEASUREMENT> is a function call to the function MEASURE which supplies

---

*A notion very similar to that used in REL (Dos71).

the quantity being measured and other arguments specifying where to measure it.

Figure 2.2

```
<MEASUREMENT> := output <MEAS/QUANT>  of <TRANSFORMER>
                 <TRANSFORMER> <MEAS/QUANT>
                 <MEAS/QUANT> between  <NODE> and <NODE>
                 <MEAS/QUANT> of <PART/SPEC>
                 <MEAS/QUANT> between output terminals
                 <MEAS/QUANT> of <JUNCTION>
                 <MEAS/QUANT> of <NODE>
                 <MEAS/QUANT> from <JUNCTION>
                 <JUNCTION/TYPE> <MEAS/QUANT>
                      of <TRANSISTOR/SPEC>
                 <TRANSISTOR/TERM/TYPE> <MEAS/QUANT>
                      of <TRANSISTOR/SPEC>
```

A careful examination of Figure 2.2 reveals that <MEASUREMENT> also accepts "meaningless" phrases such as "the power dissipation of Node 4." In addition, it accepts some meaningful phrases such as "the resistance between Node 3 and Node 14" which Sophie does not calculate at present. It is possible to write restrictions into the grammar so that phrases like these are not parsed (as they are now). In fact, we were often faced with this problem of whether or not to write the grammar to do what amounts to argument checking and to thus block meaningless parses. For example, suppose we have a function which measures the resistance of a part in the circuit and a function which measures the current flowing through either a part or a terminal of a transistor. Should the distinction between resistance and

current (i.e. that the resistance of a transistor terminal is not meaningful) be made in the parser or in the specialists which carry out the measuring? In other words, when presented with the phrase "resistance of the base of Q2", should the grammar block the parse of it? Although the above phrase would not have been parsed by earlier versions of our gramar, our tendency has been to allow the grammar to accept more statements and to have the argument checking done by the semantic routines. This has the advantage of allowing the semantic routines to provide the feed-back as to why a sentence cannot be interpreted or "understood".* It also keeps the grammar from being cluttered with special rules for blocking meaningless phrases.

The most difficult aspect of the natural language processor has been extensibility. There are basically two types of changes which must be made to the grammar. One occurs when a new feature is to be added to the system. This requires writing a new rule to accept the various ways of specifying the new feature. The other type of change involves having the grammar accept a new way of saying something that it already knows about. This requires rewriting some existing rules. It is this second type of change which usually results in undesirable interactions

---

*Providing feed-back as to why an utterance is not accepted is a difficult problem which has yet to be solved in any general way.

between rules and unforeseen side-effects. Our experience with making extensions to the grammar has shown us that it can be a difficult task but no more so than trying to add new features to any other complex program which is already written. Parts of the grammar have undergone several major restructurings and although the existing grammar represents a much larger subset of English than the original version, it is not significantly more complex nor significantly harder to change. Of course, as Sophie grows and the types of responses it must handle become more complex, there is no guarantee that the grammar won't become so complicated that new changes cannot be easily incorporated.* What we have shown is that this approach is feasible for a large enough subset of "English", and efficient enough that some natural language capabilities can be employed for CAI systems.

---

*If (when) this point is reached, Sophie will have become complex enough to employ a more powerful parsing technique (e.g. Woods' LSNLIS Parser).

Chapter 3

Endowing Sophie with Some Intelligence

Introduction

Sophie manifests most of its "intelligence" through its question answering and hypothesis evaluation and generation abilities. The primary seat of its intelligence resides in a collection of special purpose inferencing procedures each of which perform a certain class of inferences extremely efficiently. These procedures reflect an intentional move away from axiomatic deductive strategies, and although they may appear to be in the spirit of procedural deduction schemes, they are based on quite a different principle. In fact, most of our inferencing strategies are not really deductive but instead they derive a conclusion by intelligently computing "examples" and using these examples to decide on the validity of an hypothesis or the answer to a question.

By their very nature our inferencing strategies are incomplete but they are extremely skilled in answering certain classes of questions. Although we feel that we have just begun to explore the potential of this approach, this chapter presents an informal description of some of the inference mechanisms now operational in Sophie.

In addition to these mechanisms, Sophie contains numerous small "seats of knowledge" called specialists. Although only an overview is given of these specialists in this chapter, the following chapter will discuss how these specialists are organized and work in harmony with the central inferencing schemes.

Inference Generation by Simulation

The major component of our inferencing system is a simulation program which models a "piece of knowledge" which in this case is an electronic instrument*. The underlying idea of how simulation can be used to perform inferencing is straight-forward. Let us first consider the problem of evaluating a hypothesis (always with respect to a given circuit) of the form:

"If X then Y"

where X is a proposition about some component in the given instrument and Y is a proposition about its behavior or symptoms. An example of such a hypothesis might be:

"If C2 is shorted, the output voltage is zero."

The validity of the hypothesis can be tested by invoking the simulator. First the simulation model of the instrument

---

*More precisely, it models a schema of electronic instruments with one element of the schema being the working instrument and the other elements representing various ways the instrument can be faulted.

must be modified so that C2 is shorted (i.e. the proposition X must be made true on the model) and then the simulation must be executed. Since the results of the simulation run contains all the consequences of C2 being shorted, the hypothetical consequent -- the output voltage being zero -- is simply checked against these simulation results.

The above paradigm glosses over several logical difficulties concerning which boundary and/or input conditions should be used for the simulation runs. If it is necessary to determine all the logically possible consequences of a hypothetical modification, then the simulation must, in principle, be run over a potentially infinite collection of the instrument's control settings, etc. While for most practical situations there are only a finite number of cases "worth" considering, this number can still be quite large. It is clearly desirable to have an additional inferencing mechanism which can determine what the worthwhile cases are for any particular hypothesis. This additional mechanism must embody electronic knowledge of a different sort than is represented in the simulator. Thus, metaphorically, the simulator may be interpreted as creating examples whereas this additional mechanism tries to guarantee that these examples will be useful.

There are several classes of hypotheses (questions)

which explicitly dictate the control settings and boundary conditions to be used in the simulator and that can therefore directly call the simulator without needing any intelligent meditation.

Answering Questions about Particular Measurements.

The questions that fit most directly into the simulation paradigm concern requests for measurements. For example one might want to know what the voltage is across a particular component when the instrument controls are set to some values. After computing the voltage at every node in the simulated circuit, it is straightforward to derive answers to additional questions about the current through any component, the active resistance of a component, the power dissipation of a component, etc.

All these measurements can be determined either with respect to a working instrument or with respect to a faulted instrument. In this latter case the fault must be completely specified (e.g. the leakage resistance of C2 is 1000 ohms) and then used to alter the model of the circuit in the simulator. Sophie contains fault specialists who sit on top of the simulation system and are responsible for carrying out this operation. These specialists must know about how particular components can be faulted and how each component is modelled. By using this knowledge they can

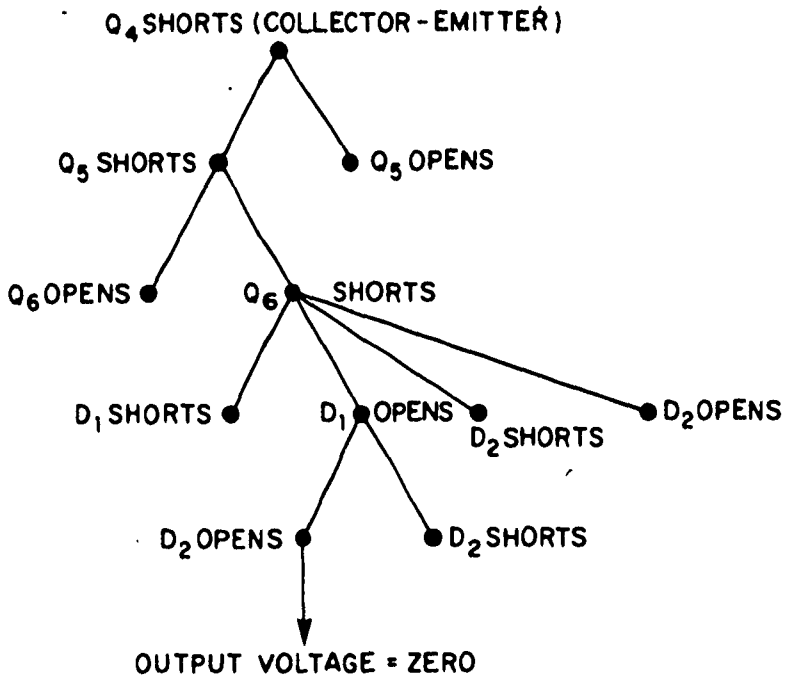
carry out the proposed modification on the internal representation of the circuit.

Inferences Invoking Fault Propagation

In addition to fault specialists there are specialists who examine the results of each simulation run in order to make sure that no component in the circuit is dissipating too much power or experiencing too high a voltage. Such considerations are important since when one component is faulted other components often become subjected to excess conditions. These specialists are responsible for detecting such situations, deciding how the given component should be blown, and then recursively calling the fault specialists to blow it in that way. At any given moment, several components might be subjected to excessive stress but if one were to blow it would protect the other from blowing. Therefore, only one component at a time is actually blown before rerunning the simulation.

These specialists are also responsible for determining an ordering on which of the overstressed components is most likely to blow. What develops is a fault propagation tree which captures either all of the reasonable consequences of a particular fault or the most likely chain of consequences.*

At each level of this tree is a set of nodes representing those component failures which are equally and most likely to occur as determined by the low level fault specialist. Then another specialist chooses one of these and re-runs the simulation. The figure below illustrates a typical fault propagation tree which was generated by faulting the IP-28 model by shorting the collector emitter of Q4.



The information in the tree is used in three inferencing tasks. The first use is in answering simple questions about a measurement in a hypothetically faulted circuit when the hypothetical fault propagates. Second, it is used in

---

* Sophie uses only this latter mode of operation.

informing the student that the component he has just replaced in his attempt to fix the instrument has just re-blown since he didn't correctly locate the root of the problem. The third use is in hypothesis evaluation where the student forms a conjecture about what is wrong and his conjecture would entail some additional component being blown.

Hypothesis Evaluation (Testing)

Another kind of inferencing task concerns the evaluation of a student's hypothesis about what component he thinks is faulted in the instrument. The evaluation must take into account the information solely derivable from the set of measurements he has taken up to the time of his conjecture. That is, it must test the logical consistency of the hypothesis against what information he should have gained from his measurements irrespective of whether the hypothesis entails the actual fault.

The evaluation strategy makes extensive use of simulation. First, the simulation model is modified so as to be consistent with the given hypothesis and then all the student's measurements are repeated under this "hypothetical" model. Next, the same set of measurements is repeated but this time in a normal or working model of the instrument. This results in three pieces of data for each

measurement that he has taken. This collection of triples is then sent to an evaluation specialist who employs various strategies for "judging" the given hypothesis and for identifying and ordering what explanatory information should be given to the student if, in fact, the hypothesis is illogical.

Hypothesis Generation

One of the most difficult tasks performed by Sophie is determining the set of possible faults that are consistent with the observed behavior of the instrument. At any time the student can ask for help and Sophie must then generate the set of hypotheses which would explain or logically follow from the particular set of measurements the student has thus far made. The way in which Sophie generates this set of hypotheses (i.e. possible faults) is to combine both backward and forward types of reasoning.

First, a backward working specialist examines an output voltage measurement (taken by the student) and generates a list of possible hypotheses that "vaguely" explain that measurement. Each hypothesis so generated is evaluated by a forward working specialist who invokes a simulation of the hypothesis to see if it really accounts for all the known output voltages and internal measurements.

In some cases the backward specialists generate as a hypothesis a fault schema, i.e. a "fault" that has some unspecified parameters. The hypothesis that "the beta of the Darlington amplifier (of the IP-28) is low" is an example of a fault schema as is the hypothesis "C2 is leaky". Rejecting such hypotheses requires some subtle reasoning: given two measurements it is possible that each measurement, by itself, could be explained by instantiating a fault schema to a particular value. However, it may turn out that the instantiations required by each measurement are in fact different and hence this fault schema cannot explain both measurements simultaneously. Although a sophisticated forward deduction system might be able to detect this inconsistency, we eventually settled on using a simulation in conjunction with a specialist who tries to find an instantiation value of the fault schema by "intelligently" manipulating the simulation model.

The exact values for these fault schemas can only be found if at least one output voltage measurement is made in which the voltage is not the correct value for the settings. However, even when a measurement is correct (in the sense of being the same value as would be found in the working circuit), it is possible to determine a range of values for these schema. For example, Sophie has a specialist who can determine the range of values for the beta of the Darlington (in the IP-28) which could account for the observed output

current. By successively refining this range, it is sometimes possible to rule out certain faults. These specialists do not use simulation but instead have enough built-in intelligence to be able to deduce these ranges for any of the fault schemas that arise in this context.

In summary, the hypothesis generation process invokes backward specialists who suggest crude hypotheses, forward specialists who manipulate a simulation model to rule out hypotheses suggested by the above mentioned mechanism, and finally, a collection of ad hoc specialists who can infer what cannot be wrong from knowing non-symptomatic measurements.

Chapter 4

Internal Design of Sophie

Introduction

This chapter provides a comprehensive description of the processes which underly Sophie. First there is a presentation of the overall organization of Sophie which includes a description of the framework within which various specialists work. Following this overview each of the specialists is discussed providing both a description of the task of each specialist as well as the method each uses to perform its task. The portions of the system which are not discussed in this chapter are: the natural language processor (Chapter 2); the general purpose simulator and the functional simulator (Chapter 5); and the semantic network (Chapter 6).

Overview

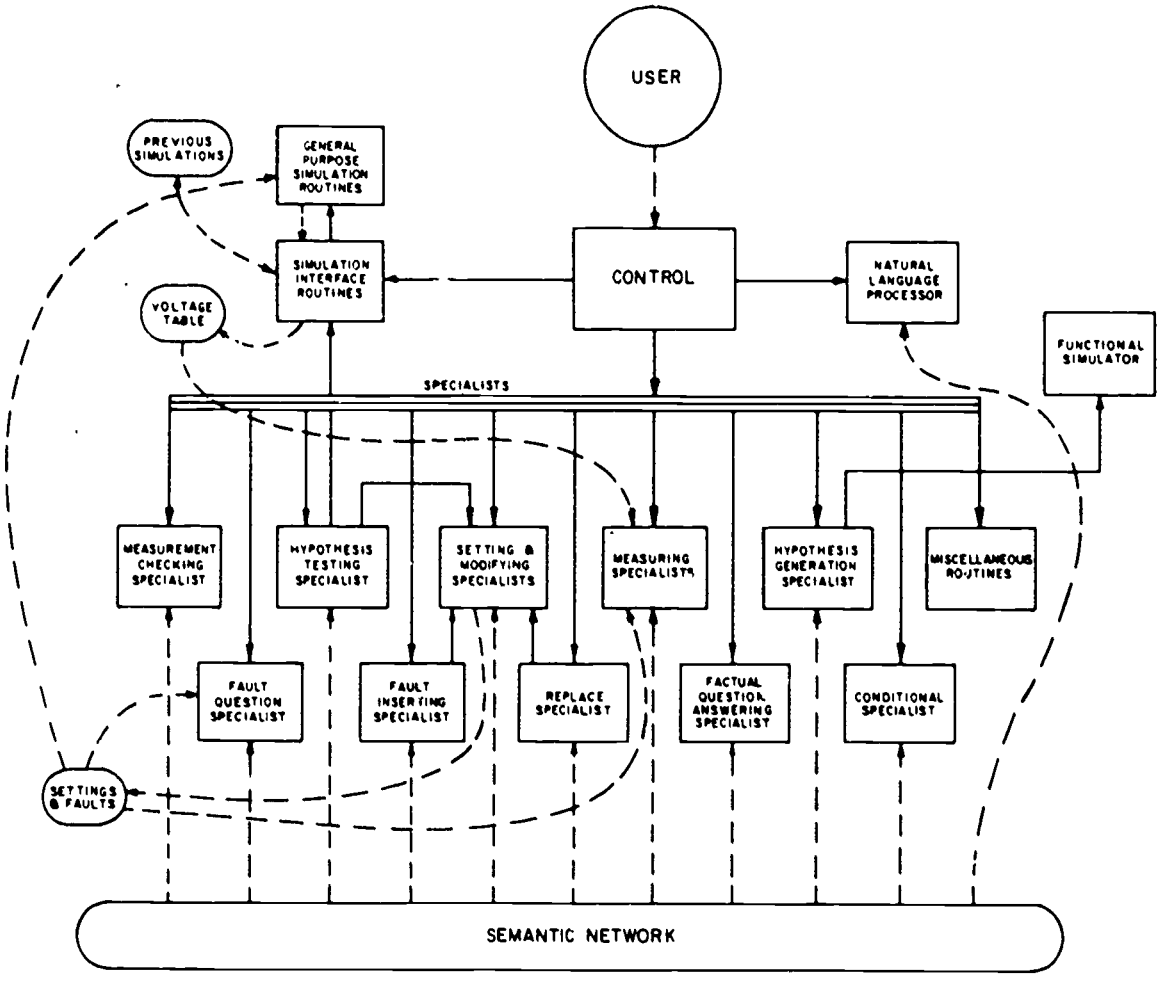
Figure 4.1 shows the basic organization of processes and data within Sophie. The processes are represented as boxes and the data areas are represented as ovals. The solid arrows indicate the flow of control* between processes

---

*Control is passed via function calls and hence always returns to the calling process. The directions of the arrows indicate which processes call which other processes.

and the dashed arrows indicate the flow of data for these processes.

Figure 4.1



Control

The processing of a statement begins when Control receives an input from the teletype. Control passes the statement to the natural language processor which parses it

and returns its semantic interpretation (see Chapter 2). This semantic interpretation is a program which represents the "meaning" of the statement. The semantic form is evaluated causing the appropriate specialist to be invoked which in turn computes an answer to the student's question or performs the student's command. It is the specialists which comprise most of Sophie's problem solving abilities. Each specialist is very good at performing specific tasks within a specific context. It is Control's responsibility to maintain the proper context for the statement at hand. After the specialist has finished, Control uses the semantic form and the result of its evaluation to synthesize a response.

In addition to calling the natural language processor, invoking the correct specialist, and constructing and printing a response, Control is also responsible for keeping track of the numerous pieces of information about what has already been deduced and what has already been presented to the student. Finding a good way to organize and update this information can drastically simplify the operation of the specialists that access it.

History List:

The information about what the student has done or has been told is kept on a history list. After each user

interaction, the type of statement, the semantic form, the result of evaluating this form and the current instrument context is saved on the history list. At any time during a session this list is a complete record of everything that has happened. The list is used by various specialists such as the hypothesis evaluation process to determine what measurements the student has taken.

Maintaining the proper instrument context:

Whenever the student requests a measurement (e.g. what is the output voltage?) there is an implied context in which that measurement should be made. Nearly all the measurement specialists operate from a voltage table which gives the voltage of each node in the circuit with respect to ground. The simulation process computes on demand this table for given control settings, load resistance and fault specification. Control is responsible for insuring that the values in the voltage table are the correct ones for answering the question currently being processed. For example, when the student changes one of the controls the voltage table must be updated to reflect this change. Whenever the instrument context is changed, Control calls the simulation interface process to set up the correct voltage table before the measurement specialist is called.

Simulation Interface Process

The simulation interface process exists between Control and the simulation process to cope with two specific problems. First, the interface process keeps track of previous runs by the simulator. When it is asked to fire up a simulation run, it first checks the list of all previous contexts to see if it has already computed a voltage table for that context. If it has, that table is returned without invoking the simulation system. Since a student is often switching between the same two states of a circuit (i.e. the faulted circuit and the non-faulted one), searching the prior contexts before running a simulation can save a tremendous amount of computer time.

The other problem handled by the simulation interface concerns what to do when the simulation is run with a fault which causes another part in the circuit to become overloaded (i.e. leading to fault propagation). To handle these cases, after the simulation program is called, specialists check all of the parts in the circuit to determine if any of them are overloaded. If any are overloaded, one possible fault is chosen and inserted into the circuit model, a message is printed to the student and the simulation is called again.* This is done so that the

---

*When the simulation is being run to check out a student's hypothesis, the process of blowing more parts is much more controlled.

student is never taking measurements in an unrealistic circuit.

Modification and Setting Specialists

The state or context of the circuit is completely determined by the control settings, the load resistance and the faults which have been introduced. This information is kept on two special variables: SETLIST which contains the information regarding controls and load resistance, and FAULTLIST which contains the information about the faults in the circuit. To change the control settings the setting specialist changes the information on SETLIST. Similarly, the modification specialist knows how to change the FAULTLIST to insert a particular fault. The modification specialists also know the fault models of the various types of parts and if given a non-instantiated fault, (C2 is leaky, the base of Q2 opens, etc.), it will propose a complete fault to the student and interact with him until a complete fault has been agreed upon.

Measurement Specialists

The specialists which calculate voltages, currents, power, etc. know how to use the voltage table to answer the questions at hand. For example, VOLTAGE?, the specialist which calculates voltage measurements, can measure the

voltage across a part, across the load resistor, across a junction of a transistor, between two points in the circuit or at a single point with respect to ground. Before using the voltage table, VOLTAGE? accesses the semantic net which contains, for example, the information required to map between terminals and their locations in the table of voltages. In addition to VOLTAGE?, there are the CURRENT?, RESISTANCE?, and POWER? specialists which measure the obvious quantities. Current measurements can be rather complicated when using just a table of voltages as in the case of the current through the terminal of a transistor.

The different types of measuring specialists are all called by a single routine MEASURE which is returned in the semantic form from the parser. This allows MEASURE to perform some argument checking which would otherwise have to be done separately by each of the other routines. In addition, by using a relatively small number of top level semantic routines, the information on the history list is more easily deciphered by the various processes which use it.

Answering Factual Questions

One type of factual question which Sophie can answer deals with the various properties of parts in the circuit. This information, such as the beta of a transistor, is

stored in the semantic net and is retrieved by the process CHECKSPEC. Another type of factual question concerns information which is not time invariant and therefore not stored in the net. This type of question is considered below.

Fault Questioning Specialist

In certain modes of use, it is permissible for the student to ask whether a particular part is faulted. (This facility is especially useful when troubleshooting faults which cause many other parts to blow.) These requests are handled by the fault questioning specialists (SPEFAULT). The context needed by this specialist is the list of faults and using this list, this specialist can determine i) if a particular part is faulted a particular way (e.g. "Is R9 open?", "Is the leakage resistance of C2 5000 ohms?," "Is the beta of Q5 too high?", "Is the collector of Q3 open?"); ii) if a particular part is faulted in any way (e.g. "Is Q5 bad?"); or iii) if any part is faulted at all (e.g. "Is anything wrong?"). While answering some of these questions may require little more than a quick check of the list of faults, (e.g. "Is anything wrong?"), some require additional processing. For example, to determine if a particular junction of a transistor is bad requires first discovering if the transistor involved is faulted and, if it is, decoding its fault mode to see if it entails that

junction. Also, to determine if a property of a part is too low requires not only finding out what the value of that property is but also what it should be.

Inserting Faults

The specialist which inserts faults into the circuit (INSERTFAULT) is called when the user asks Sophie to introduce a fault into the circuit. It is this specialist's job to decide which fault should be inserted. (After a fault is chosen, the modification commands are called to make the appropriate changes to FAULTLIST.) Faults are known to the system by number and it is possible that the student asked for a particular fault by its number. (His instructor may have given him the number.) In those cases when the student did not ask for a particular fault, he may still have specified a modifier of some kind (e.g. hard or easy). The inserting routines keep lists of the faults which satisfy the various modifiers. Another list records which faults have already been inserted so that the student does not get the same fault inserted twice in one session. Also, before the modification routines are invoked the circuit is cleared of any previous faults. After the circuit has been modified, the semantic network is checked for any special instructions about how to handle this fault.

Replacing Parts

When the student asks that a part be replaced, the replacement specialist (REPLACE) queries the student as to how the particular component is faulted (open, shorted, base open, etc.). Requiring the student to know the particular fault mode of a part encourages him to think more deeply about the symptoms he has observed. Determining how a component is faulted rather than just that it is faulted often requires a second examination of the chain of effects which led to discovering the problem. If the part is not faulted in the way in which the student has said, he is told so and the part is not replaced. (This should not be confused with the action of the hypothesis testing routines whose job is to determine if what the student thinks is wrong is consistent with what he knows.)

There are times when the system should be aware that the student already knows what is wrong with a part and then it should not question him. For example, when the student tells the system to modify a given component, he shouldn't be queried when he wants it replaced.* The student also should know what the fault is after he has received an affirmative answer to a question such as "Is R9 open?".

---

*This may not always be the case: the instructor may have inserted the fault and then turned the system over to the student. Consequently a flag can override this situation.

Information that the system thinks that the student knows about faults is kept on the list USERFAULTKNOWLEDGE. Before the student is asked about the fault, the replacement specialist checks this list.

The questions asked a user are determined by the type of part being replaced. For each type of part (transistor, resistor, diode, capacitor, etc.) there is a corresponding specialist which knows how that component can be faulted. Once this specialist has completely determined (via questioning) what the user thinks is wrong, it calls the fault questioning specialist to see if that fault is in fact present. In the case of a transistor, the part is replaced if the student is correct about at least one of the junctions.

Measurement Checking Specialist

While troubleshooting a fault, a student may need to know whether a measurement that he has just taken is correct,* that is, is it a symptom. Sophie allows the student, after a measurement has been taken, to ask if his previous measurement was correct, e.g. "Is THAT right?". To handle this request the measurement checking specialist first gets from the history list the semantic interpretation

---

*"Correct" meaning the same as it would be in a non-faulted circuit.

of the last question asked by the student. The specialist then sets up the measurement taking environment of a working circuit by clearing the fault list and calling the simulation interface specialist to determine the correct table of voltages for the non-faulted circuit under the present control settings. The previous semantic form (measurement) is then EVALuated in the newly established context to determine what the measurement would be in a working circuit under the same control settings. The student is told what the correct measurement is and this value is compared with the value he observed earlier to determine its correctness. Then the original fault list and table of voltages are restored, thereby restoring the context of the faulted circuit.

Hypothesis Testing Specialist

The purpose of the hypothesis testing specialist is to provide Sophie with a way of checking the logical consistency of the student's conclusions. It does this by taking the student's proposed fault and determining if the symptoms of that fault agree with the symptoms that the student has observed. If there is agreement in all of the measurements that the student has taken, the proposed hypothesis is consistent. If the results of the proposed fault disagree at some measurement, this measurement represents either a consequence of the circuit's operation

which the student overlooked or was unaware of, or it represents an error in the student's logical thinking processes.* Sophie has the capability of finding such points of disagreement and telling the student about them.

The hypothesis testing specialists are invoked when the student proposes a hypothesis (e.g. "I think that C2 is leaky" or "Could the problem be that the base of Q3 is open?"). Because these specialists derive all of their inferences by inserting a fault into the circuit, they must know exactly which fault the student intends. So if the student proposes a fault schema (e.g. C2 is leaky but what is its leakage resistance), these specialists propose a reasonable instantiation of the schema but allow the student to override it and provide his own value. Once the complete fault is known, the list of faults is checked to see if the hypothesized fault is, in fact, what is wrong with the circuit. If so, Sophie informs the student that his proposed hypothesis is consistent with all of his measurements.

To evaluate a hypothesis which is not the real fault, the hypothesis specialists must know all of the measurements that the student has taken under various instrument

---

*If this is the case, the student can explore the proposed fault immediately using the save-restore facility.

settings. This information is obtained by scanning the history list. The measurements that were made are divided into two classes: external and internal. The external measurements are the measurements a student could take in a real electronics laboratory without taking the cover off the instrument, that is, the output voltage and output current. The internal measurements are everything else. The internal measurements are further broken down into context frames, that is, all of the measurements taken under the same control settings are grouped together.

The specialist then checks the consistency of the measurements one context frame at a time. Within a frame, comparisons are made between each of the measurements and what that result would be under the proposed hypothesis. The comparison requires knowing not only the measurement in the hypothetical case but the one in the working circuit as well. The working measurement is necessary to determine when the other two measurements differ significantly. For example, if the measurement that the student observed was 25 volts and the measurement under the hypothesized fault was 30 volts the difference between these two may or may not be significant. If the measurement in the working circuit is 30 volts, the proposed fault does not account for the lower voltage observed in the faulted circuit. However, if the working circuit voltage is 3 volts, the hypothesis is doing a pretty good job of explaining this measurement.

The order in which Sophie comments upon the discrepencies that it finds can be critical. The hypothesis testing specialist first comments on all of the measurements which the student has made in the latest context. Since the simulation under the various contexts is saved, this insures that the student will get some feedback within at most, two simulation runs thereby guaranteeing a minimum response delay. This feedback will include those measurements which he has made most recently and are foremost in his mind. The second group of measurements commented upon are the external measurements. Then the measurements from each of the other frames starting with the second most recent are discussed.

Each time comparisons are made within a group of measurements the results of these comparisons are reported to the student and he is asked if he wishes the evaluation procedure to continue. In reporting the results of the comparison, the measurements are ordered according to the amount of disagreement between the observed measurement and the hypothetical value. For each measurement there are four cases that might occur. The observed and hypothetical values may agree. The observed value may represent a symptom (i.e. be wrong) while the hypothesized value is right. In this case, the fault proposed by the student does not account for this particular symptom. The observed value may be correct while the hypothetical value is wrong. In this case the proposed fault would create symptoms which the

student did not observe. Or the observed result and the hypothetical result may both be wrong but not the same. In every case but the first, the student is told how the measurements disagree and is given the observed, the hypothesized and the correct value of his measurement. If no differences are found, the student is told this also.

A necessary prerequisite for performing the above procedures is the ability to determine when the hypothesized measurement and the observed measurement disagree significantly. The specialist which makes this decision uses three pieces of information: the values of the hypothesized measurement, the observed measurement and the working measurement. It calculates a "distance" between the hypothetical value and the observed value. This distance is basically a product of two factors. One factor is the percentage difference between the two values. The other factor is based on how far these two values are from the correct value.

A problem occurs when a simulation with the hypothetical fault reveals that it would cause other components to blow, that is, when the fault propagates. In this situation, the overloaded parts are not automatically blown. Instead, the student is informed that the fault he proposed would propogate and that new faults are likely to occur. The most likely fault is chosen and the student is

asked if he would like the evaluation of his hypothesis to continue with the additional fault. This gives the student a chance to avoid having to wait while the simulation runs again while still providing him useful feedback about his hypothesis.

Conditional Specialist

The conditional specialist is invoked to answer if-then types of questions (e.g. "If C2 shorts what happens to the current thru the collector of Q5?"). As was stated in Chapter 3, the method the conditional specialist uses to answer this question is to create a context in which C2 is shorted and then to call the measurement specialist to determine the collector current of Q5. In addition to putting a particular fault in the context that it establishes, the conditional specialist must also decide what control settings and load resistance to use. The strategy employed is to first try the present settings. If under the present settings, the consequent measurement is the same in both the faulted circuit and a working circuit, then these settings are probably not the proper ones to demonstrate the effects of the fault and the conditional specialist checks the semantic network for more appropriate settings. In any event, the student is told what the consequent measurement is in both the working and faulted circuits.

Hypothesis Generation Specialist

When a student, after taking several measurements, cannot think of any possible faults which would explain those measurements, he can ask for help. The hypothesis generation specialists have the job of providing the student with some possible faults. Chapter 3 describes the method used by this specialist.

Miscellaneous Routines

In addition to the major types of features described earlier, Sophie offers several other features. One of these (REVIEW) provides the student with a review of all of the measurements he has made since the beginning of a troubleshooting session. This can be very useful in long sessions or when using a CRT terminal.

To save the initial cost of starting Sophie, there is a restart command which re-initializes the system but does not re-perform the expensive operations of setting up the simulation process. A similar operation is provided by the SAVE and RESTORE commands. The SAVE command saves the state of the current session so that it may be returned to by a RESTORE command and performs a recursive call on Sophie. This is mainly useful during a troubleshooting session after the student has been told that one of his hypotheses is not

completely correct. The student can then immediately explore his hypothetical fault by SAVEing the current session and inserting his hypothetical fault into the circuit of the new version of Sophie. When he has completed his exploration, he can RESTORE his trouble-shooting session and continue where he left off, trying to find the real fault.

Chapter 5

Simulation Techniques

Introduction

As we indicated in Chapter 3, most of our inferencing strategies are centered around the use of simulation models. This chapter describes the two kinds of simulations used in Sophie. The first is a general purpose simulation system which accepts a description of an arbitrary circuit and produces exact quantitative results in both working and faulted version of the circuit. This system is used to answer all requests concerning measurements. It is also used by the hypothesis evaluator.

The second simulation system is a circuit-dependent, functional simulator which runs several orders of magnitude faster than the general purpose one but which produces only approximate results. This system incorporates much specialized knowledge about the internal functioning of the IP-28 instrument and currently is used only by the generation specialist.*

A functional simulator which handles the class of all faulted instruments stemming from one working instrument is

---

*This is the section of Sophie which is the most difficult to change when giving it a new instrument to model.

considerably more complicated than a functional simulator for just the working instrument. In particular the "transfer function" of each functional block must be so modelled that any internal component of a block which becomes faulted can be directly translated into a new transfer function. Also if the simulator takes advantage of the internal logical constraints inherent in the instrument, it must be able to take into account all the ways these constraints can be influenced by any possible fault. Nevertheless a highly tuned functional simulation of an instrument can provide such significant speed-up that uses of simulations which were previously unthinkable can now be quite effective.

The General Purpose Circuit Simulator

SPICE* (Nag71)(Nag73) is the general purpose electronic simulation program used in our system. The input files for SPICE contain a description of the topological arrangement of the circuit components and their nominal values; specifications of parameters for whatever models may be used for non-linear components such as BJT's and diodes; and a

---

*SPICE was originally written by Laurence Nagel at the University of California at Berkeley. The original program consists of some 8,000 lines of FORTRAN source code and was designed to run on the CDC 6400.

list of desired output options. Possible circuit components include independent and dependent voltage and current sources, field effect transistors, BJT's and diodes, as well as the usual linear devices. Provision is also made for specifying model devices not already included in SPICE's basic library.

SPICE performs three different types of circuit analysis: DC operating point, AC response spectrum, and transient behavior. SPICE's DC operating point analysis includes operating point values for model parameters as well as the voltages at each node of the circuit. Transient analysis consists of a series of DC analyses against a varying input voltage, with charge storage elements taken into account. The AC analysis subprogram simply determines the DC operating points whose values are phasors rather than real numbers. AC solutions are expressed separately for the given input signal frequencies rather than as dependent functions of the frequency. Since AC analysis assumes that all nonlinear devices operate linearly sufficiently close to a particular DC operating point, it is of limited use for deducing if a particular modification has caused a component to instantaneously enter a saturation or cut-off state, etc. In addition, since the CPU time required for transient analysis is prohibitive (on the order of ten CPU-seconds for the circuit under consideration), we use only the DC operating point analysis.

DC Analysis Package

The DC analysis subprogram is divided into three sections: READIN, SETUP, and ITERATION. READIN accepts the description of the circuit and inserts this information into the various internal arrays. Array-stored information includes the nodes to which each element is connected, the value of each element, and the type of each component. READIN also simplifies the external description (e.g. it re-numbers the nodes so that they are consecutive) and checks for possible specification errors.

SETUP allocates space in the sparse array and stores the pointers to it. It first checks each component and determines which locations of the array will be occupied. It then adds whatever locations are required for forming intersections with other locations already allocated. Finally, SETUP stores the appropriate pointers for each component.

The ITERATION section solves the internal representation created by READIN and SETUP. The method of solution is based on Kirchoff's and Ohm's Laws. The basic matrix equation is:

$$E = Y^{-1} I$$

where I is the vector of source currents for the nodes, Y is the matrix of mutual conductances between nodes, and E is the resultant vector of node voltages. The Y matrix is

"loaded" by adding the conductance of each component to the two diagonal terms and subtracting it from the off-diagonal ones. The I vector is then loaded with the values of the known current sources. After these are loaded for all the circuit components, the Y matrix is inverted by the Doolittle method (Fox65) and multiplied by the I vector. This obtains the voltage for each node.

The above method is all that would be required if the circuit contained only elements with linear I-V curves. However, transistors and diodes have exponential curves. These non-linear characteristics are handled by an iterative process. A "best guess" of the voltage drop across a diode, for example, is computed and used to determine a point on the I-V curve. The inverse of the derivative at that point is then determined and used as the value of the effective conductance, thus modifying the Y matrix. The I intercept of this derivative is then determined and used to modify the I vector. The new Y matrix is then inverted and the basic matrix equation is re-solved for the new values. The resulting E vector is then used to replace the previous "best guess" and the above sequence of events is repeated. This process may be thought of as forming a Taylor series approximation to the non-linear I-V curve. After each iteration the newly obtained voltages are compared with those produced by the previous iteration. If the voltages are within Ø.1% or 5Ø microvolts of each other, whichever is

greater, then the analysis is complete.

In certain circuits, the iterative process will fail to converge. If 100 iterations are completed without convergence the analysis is stopped. Fortunately this happens primarily with bistable circuits, a situation that should not arise in our case. Also, good convergence may not be achieved due to the rapidily changing nature of the exponential equation. To speed up convergence by limiting overshoot, a limit of 1.4 volts is placed on the amount of change in the voltage across semiconductor junctions.

Modifications Made to SPICE

In addition to converting SPICE to run under DEC's Fortran, we removed all code which dealt solely with AC and transient analysis. We also removed the ability to handle component models, like those for MOSFET transistors which are not likely to be used in the circuits being simulated. The size of the matrices was reduced since the branch and node capacities of the program were greater than required. This modified program is approximately 3500 lines of FORTRAN code.

Introducing Faults into SPICE

The major addition to SPICE is the introduction of a mechanism for introducing faults and circuit modifications

quickly. One method of achieving this would be to modify the external description of the circuit and read it in each time the effect of a fault is sought. In the case of a transistor with an open junction, for example, the input statements to SPICE would be changed to show a diode for the good junction and an open circuit for the defective one. The circuit description would then be analyzed in the manner described above. It was felt, however, that this would be too slow.

The method chosen instead was to change the models of all the components so that they take an additional piece of information from a fault array. Actually two arrays are involved: IFAULT is used to tell how a component is faulted and FAULT is used to hold the faulted value of the component. For example, the first array might tell if a resistor were ok, open, shorted, or had changed its value. The second array would only be used in the case that the value had changed and would then be used to contain the new value.

The actual faulting is done when the component is being "loaded" into the sparse matrix. For example, the routine for a resistor normally looks at the array VALUE, which was setup when the initial circuit was read in. This value is the conductance of the resistor. It would then add this value to two locations and subtract it from two other

locations in the sparse matrix. However, if IFAULT indicates that the resistor is not normal, the routine uses the value in FAULT when the resistor changed value. Otherwise, it uses conductances (inverse of resistance) of 1ØØ. and 1.ØE-1Ø to represent shorted and open resistors, respectively.

The diode and transistor models have series resistors on their terminals. Opening a diode is performed by making this conductance 1.ØE-1Ø. Shorting the diode is more difficult. There exists four locations in the sparse array for the diode. As with resistors, these locations are modified by 1ØØ to indicate the junction is shorted. Transistors are similar to diodes. If a lead is to be opened, the series resistor is opened. There are four locations in the sparse array for each junction of the transistor. Two of these are shared with one of the other junctions. As with the diode, these four locations are modified by 1ØØ. Changes in beta are performed by substituting the beta in FAULT for the normal beta when loading the transistor.

Performance of General Purpose Simulation on DC Analysis

For a faultable model of the IP-28 circuit (see Appendix 1) SPICE takes on the average 1.8 cpu seconds (PDP-1Ø). The faultable model actually contains twice as

many nodes as a simple model of the IP-28 because of the additional parts needed to allow certain fault modes.

The Circuit-Dependent Functional Simulator

One of the practical limitations to simulation concerns the issue of speed. Although certain questions might require only one simulation run, other kinds of questions require many runs and in some cases just one simulation can involve several CPU seconds. What was needed was an additional hierarchical simulation model which could take advantage of specialized knowledge about a schema of circuits. In particular it should be hierarchical in the sense of simulating functional blocks and should utilize logical constraints which hold over these functional blocks to intelligently guess the effects of various feedback loops, etc. It should also be designed to facilitate the introduction and handling of a wide class of faults pertinent to each functional block.

A circuit dependent simulation can be made which is between 100 to 1000 times faster than the general purpose simulation and thereby opens up an entirely new dimension of uses. This speed-up has been gained at the cost of generality by building into the system detailed knowledge about the purpose and characteristics of each function block. Speed has also been gained at the expense of

accuracy. For example, the simulator does not model the exponential curve of a transistor but instead "deduces" if a transistor is on and if so, assumes its VBE is about .7 volts, etc.

Modelling new instruments requires a thorough logical analysis of their internal states under faulted conditions as well as when properly working. Although this analysis might be automated,* at this point in our research, it is being done by hand.

Using Circuit Dependent Knowledge in Modelling the IP-28

The purpose of the following subsections is to provide an indication of how circuit dependent knowledge can be used in creating a fast simulation which has enough flexibility to permit the insertion of a wide class of faults. If the reader is not already familar with the internal workings of the IP-28 power supply, he is referred to Appendix 1.

Functional Simulation of the IP-28:

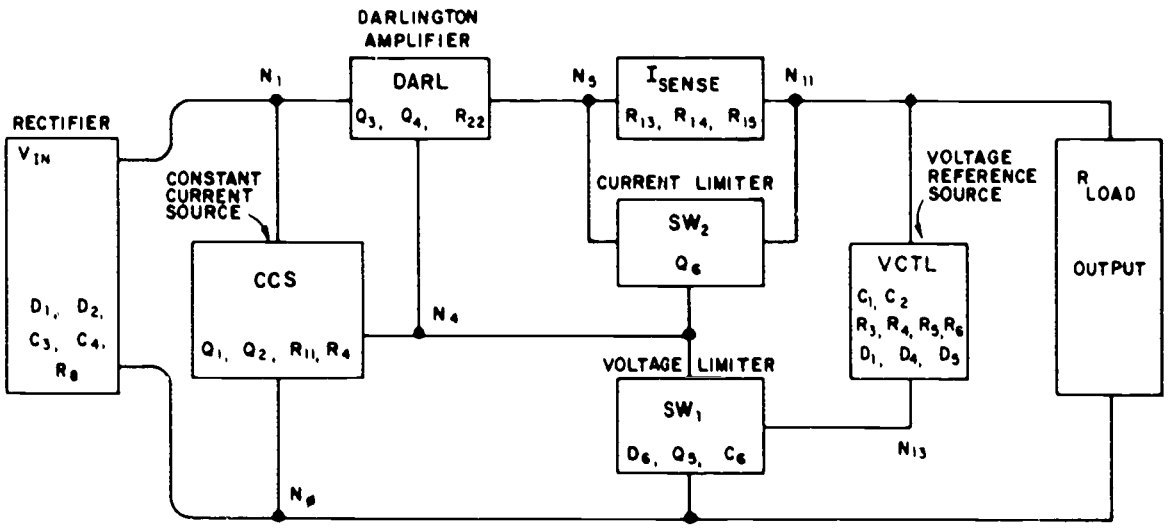
This simulation model consists of eight functional blocks whose interconnections and mnemonics are illustrated

---

* A novel use for Sophie would be to aid in constructing this analysis. After a general purpose simulation of a new circuit is given to Sophie, the interactions of the functional blocks could be explored under various fault conditions.

in Figure 5.1.

Figure 5.1



The transfer function of each of these blocks is first encoded as a LISP procedure. The following algorithm is then used to determine the behavior of these interconnections. First the output voltage is computed under five different assumptions -- each assumption leading to a possibly different output voltage:

Assumption 1: The output voltage is being determined (limited) by the voltage control setting.

Assumption 2: The output voltage is being determined (limited) by the current control setting.

Assumption 3: The output voltage is being determined (limited) by the amount of current capable of being delivered by the Darlington.

Assumption 4: The output voltage is being determined (limited) by the maximum voltage produced by the constant current source.

Assumption 5: The output voltage is being determined (limited) by the voltage produced by the power supply (VIN).

(The last three assumptions are pertinent only when we are dealing with a faulted circuit.)

The simulation algorithm picks the lowest of these five voltages and then uses this voltage to calculate such quantities as the voltage drop across ISEN, the output current, the amount of the CCS current being dissipated by SW1 and SW2, and the voltage drop across the base emitter of the Darlington. However, some of the five output voltages computed above depend on the value of these internal quantities (e.g. the voltage being produced by VIN depends on the output current) and hence after these are computed the five output voltages are re-computed and the whole process re-iterated until the process converges (i.e. 1% variation between two successive iterations). Usually, this process converges in three to four iterations.

Simulating faults in the model: 08-

There are two kinds of faults each of which is handled differently. The first kind is a catastrophic fault such as SW1 being shorted which preempts the above simulation and immediately dictates the resulting behavior -- the output voltage is zero.

The other kind of fault invokes the simulation to determine its effect. With these non-catastrophic faults the resulting behavior can be captured by changing parameters or ignoring certain sections. Since the beta of DARL is used to calculate one of the 5 voltages, changing this parameter would change that voltage and possibly the output voltage. Likewise, if SW1 were open, the simulation would ignore the voltage produced by the voltage control and would take only the minimum of four voltages. A third example would occur if D6 were shorted. In this case, the voltage produced by the VCTL would be different by Ø.7 volts.

Embedding faults in the function blocks:

The mapping between specified faults and the information used by the simulation is done in two ways. The first is pertinent to the VIN and CCS function blocks and consists of directly modifying their transfer functions. This is necessary since their behavior is dynamic with

respect to some condition which varies during the simulation. For example, VIN needs to know the output current to calculate its voltage. CCS needs to know what voltage is across it.

The second method uses functions which are evaluated before the simulation starts and which produce attribute lists which are useful in determining the behavior and effect of the block under a particular fault. For example, the attributes for SW1 are: whether SW1 has any effect on the output voltage, whether the Darlington is off because node 4 is shorted to ground, the beta of SW1, the status of D6, and whether the BE junction is shorted. The simulation system uses these lists in order to determine if this block can effect the output voltage and also to determine various internal measurements. The attributes for DARL are: the beta, the BE voltage drop and the value of R22. The attributes for SW2 are: whether SW2 has any effect, the voltage drop across the BE, and whether DARL is forced off. The attributes for VCTL are: the voltage and resistance. Finally, the attributes for ISEN and RLOAD are their resistances.

Most of the above knowledge is quite ad hoc but is extremely useful in constructing a fast simulation model which can also simulate realistic faults.

Chapter 6

Semantic Net

Introduction

The semantic network provides a uniform yet efficient and very general means of representing the many different types of non-procedural time-invariant knowledge required by Sophie.* Developing specialized representations for each type of information would be more efficient in terms of size and speed but would be much less efficient in terms of the programming effort required to make changes and additions. Since our purpose was to experiment with different approaches to representing and utilizing semantic/conceptual information, we chose a completely general form of a semantic net -- one in which we could freely modify and extend our use of the semantic net without fear of precluding some kind of retrieval or processing strategy.

Implementation of the Semantic Net

The modelling structure of the semantic net is a descendant of Shapiro's semantic network of the MIND

---

* The net is not used for storing information about the sequences of measurements made by a student. For such information a tailored data structure is used.

system (Sha71). The network consists of conceptual entities or "items" connected by relations which hold between the items. An item can be thought of as anything (object or concept) about which information is known. Some examples of items are D6, the class of diodes and the concept of D6 shorting. The network implementation is compatible with the semantic network used by the speech understanding project at BBN (Woo74)*. It includes convenient ways to add new information to the net, delete or change relations in the net and, of course, retrieve information from the net. There are also functions for printing items in the net, displaying the entire network, storing the net on disk, and expanding and contracting the net. The implementation provides the user with a complete set of LISP functions for constructing, editing and retrieving information from a general semantic network which is, in principle, capable of representing any non-procedural information.

Shapiro's notion of a semantic network has been extended to allow an item (conceptual entity) to have properties as well as relations. The distinction between a property and a relation is that a property is a relationship

---

*We are indebted to Ron Kaplan and Bonnie Nash-Webber for sharing with us some of their code. Although our decision to use this implementation structure was made independent of the speech project, it hopefully will enable us to utilize more of their linguistic research in building still more powerful CAI systems.

between an item in the network and a structure outside the net whereas a relation can exist only between two items in the net.* Properties allow us to make limited use of specialized representations outside the network for restricted pieces of information. For example, the fact that C2 is a capacitor is represented by a relation link (MEMBER/OF) because both C2 and capacitor are represented by items in the net, but the fact that the value of C2 is 5Ø Microfarads is represented by a property (VALUE) because "5Ø Microfarads" is not represented in the network. There is no a priori reason why "5Ø Microfarads" could not be an item but since it stands in no relationships of interest with any other item in the system it is more efficiently represented as a property (i.e. the system is not interested in structures over units or numbers).

Whenever a relation is added between two items, the inverse relation between these two items is automatically added.** For example, we can add the information that D6 is a member of the voltage limiting section and is between Q5 and R7 by using the function ADDITEM as follows: (ADDITEM D6 (PART/OF voltage/limiting/section) (CONNECTED/TO Q5 R7)). The effect in the net is to put a PART/OF link from item D6

---

*This is a moot point for Shapiro who represents all information in the net.

** Note this is not true of properties which have no inverses.

to the item voltage/limiting/section and also a HAS/PART link (inverse of PART/OF) from voltage/limiting/section to D6. In addition there would be a CONNECTED/TO link from item D6 to both items Q5 and R7 and from Q6 and R7 to D6 (CONNECTED/TO is its own inverse). This means that the part of the network dealing with relations is a two-way graph which allows a simple, very powerful accessing method which can perform certain classes of inferences.

Retrieving Information from the Net

The basic accessing function for relational information in the semantic net is the function IFIND. IFIND retrieves items from a specification of how those items are connected to other items. The specification is given in terms of relation-item/list pairs of the form (R (I1 I2 ...In)) which give the relations (R) and the items (Ik) which constrain the desired items. Given a list of relation-item/list pairs, ((R1 (I11 ... I1p))(R2 (I21 ... I2r))...(Rn (In1 ... Inq))), IFIND returns all of those items in the network which for all Rk stand in relationship Rk to at least one of the Ikj. In the simplest case there is only one relation-item/list pair and the item/list contains a single item. For example, (IFIND (PART/OF VOLTAGE/LIMITING/SECTION)) will return a list of those items which are parts of the voltage/limiting/section that is, those items which stand in relationship PART/OF to the item

VOLTAGE/LIMITING/SECTION. This list is easy to find since all of the desired items are connected to voltage/limiting/section via a HAS/PART relation (the inverse of PART/OF). In the relation-item/list pairs, the item/list can be specified as calls to IFIND which allows IFIND to be nested to any depth. Using our earlier example again, (IFIND (CONNECTED/TO (IFIND (PART/OF VOLTAGE/REFERENCE/SECTION)))) will first find all of the items which are parts of the voltage/reference/section and then return the union of those items which are "connected to" any of the items on this list.

In the case where there is more than one relation-item/list pair, IFIND takes the intersection of the lists returned by the single pair case. Thus,

```
(IFIND (PART/OF VOLTAGE/LIMITING/SECTION)
 (CONNECTED/TO (IFIND (PART/OF VOLTAGE/REFERENCE/SECTION))))
```

will find those parts in the voltage limiting section which are connected to the voltage reference section. The process goes as follows. First form a list of all of those items which have a PART/OF relation with voltage/limiting/section. Next form a list of all of those items which have a CONNECTED/TO relation with any item which has a PART/OF relation to the voltage/reference/section. Note the amount of redundant searching which is avoided by forming this second list since each part in the voltage reference section is on this list only once but may be reachable via many

different CONNECT/TO links. The answer to the request is those items which appear on both of these lists. Thus IFIND works by taking the intersections of unions of lists of items. Since retrieval of this kind is potentially costly, the storage structure of the network is maintained in sorted order to allow efficient implementation of the intersection and union operations.

An Example of the Net

Figure 6.1 shows several items from the semantic network (as they appear when displayed by DESCRIBE, one of the network printing functions). The items shown are: item 1, capacitor; item 3, the particular capacitor, C2; item 9, the fault F2 which is a part failing by shorting; and item 82, the concept of the particular capacitor C2 failing in the particular way F2 (shorting).

To give an idea of the way information is represented, we will give a description of the item representing C2. PNAME is a special property which links the network (which is implemented as an array) to LISP atoms. On the property list of the atom C2 is the property SREF and the value 3 (this item's array location) which means that the C2 provides an entry point into the net.* VALUE, BREAK/DOWN/VOLTAGE and LEAKAGE/RESISTANCE are properties which C2 has. TOO/LOW/REST is the resistance that is

Figure 6.1

Examples of Items from the Semantic Network

```
ITEMS:

1
   PNAME            CAPACITOR
   ARG/TO           (F1) (F2) (F4)
   MEMBER           (C1) (C2) (C3) (C4) (C5) (C6)
   MEMBER/OF        (COMPONENT)

3
   PNAME            C2
   VALUE            (5Ø MICROFARADS)
   BREAK-DOWN/VOLTAGE
                    (5Ø VOLTS)
   LEAKAGE/RESISTANCE
                    (1ØØØØ MEGAOHMS)
   TOO/LOW/REST
                    (1ØØØØ OHMS)
   HAS/NEGATIVE/TERM
                    (N/C2)
   HAS/POSITIVE/TERM
                    (P/C2)
   HAS/TERMINAL
                    (P/C2) (N/C2)
   MEMBER/OF        (CAPACITOR)
   PART/OF          (REFERENCE/VOLTAGE)
   PARTPR           81 82 83

9
   PNAME            F2
   EGO              (SHORTED)
   ARG              (CAPACITOR)(DIODE)(RESISTOR)(SWITCH)
                    (TRANSFORMER/WINDING)(ZENER/DIODE)
   FAULTPR          75 82 9Ø 93 96 98 1Ø4 1Ø6 1Ø8 115 185 19Ø
                    194 2ØØ 2Ø5 2Ø7 2Ø9 213 217 221 226 228
                    232 237 243 248 25Ø 252 254 256 258 26Ø
                    262 264 267
   MEMBER/OF        (FAULT)

82
   FAULT            (F2)
   PART             (C2)
   SCHEMA           (S33112)
   FAULT-SETTINGS   (LOAD 2Ø) (CC 1.Ø) (VC 1.Ø) (CR HIGH)
                    (VR HIGH)
```

proposed when a leaky C2 is inserted into the circuit. The user can state a specific value for the leakage resistance if he so desires; this value is the default when none is specified.

HAS/NEGATIVE/TERM and HAS/POSITIVE/TERM are respectively the normally negative and normally positive terminals of C2. These are used to provide the proper sign when the user asks for the voltage across C2, i.e. in a normal circuit the answer will be positive, but if the circuit is faulted in such a way that the potential difference is reversed, the sign will be negative. The HAS/TERMINAL is used in conjunction with TERMINAL/OF/NODE links to represent the topology of the circuit. Note that this information is a duplication of the HAS/NEGATIVE/TERM and HAS/POSITIVE/TERM links. This duplication is necessary because the implementation of the semantic network does not allow structuring over relations. This is a shortcoming of our semantic net and is a time-space trade off in favor of time.

---

*In fact, atoms with SREF properties (terms) are the only way to access the net. Note that this allows the network to have items which do not have associated atoms. This can amount to a considerable savings over a net implementation based on atom structures. In the electronics network of 450 items only about one-fourth of these need atoms associated with them. (However most of the rest do have a PNAME but these are only for debugging purposes and could be removed in a finished system.)

The MEMBER/OF relation represents the fact that C2 is a capacitor. The PART/OF relation indicates which section of the circuit C2 is part of. Using these last two relations we can determine that C2 is a voltage reference capacitor. PARTPR are links to concepts of C2 failing in specific ways. Item 81 represents the concept of C2 opening, 82 that of C2 shorting and 83 the concept of C2 being leaky.

Network Functions*

The items in the semantic net are represented by small integers which point to cells in the network storage area, the array ITEMARRAY. Each cell of array storage can hold two LISP pointers. The CAR of an array cell is an ASSOC-type list of the relations of that item while the CDR is its property list. To initialize a network, there is a function NEWNET which takes one argument, the maximum number of items. NEWNET allocates an array of proper size and assigns it to the global variable ITEMARRAY. It also clears the global variables TERMS (a list of the atoms which correspond to items), EGO-DICT (a list of item print names), and FREENODE (the free list of items). Whenever a new item cell is desired, the function NEWITEM is called. If NEWITEM

---

*This section is a rather detailed description of the set of network functions and would be of interest primarily to those wanting to use this set of functions as tools for building their own networks.

cannot find a free item cell in ITEMARRAY, the function EXPANDNET is called to automatically double the size of the storage array.

Before adding any information to the net, the user must define the relations and properties he intends to use. This is done with the function DEFRELS, e.g. (DEFRELS (R1 m/s Rlinverse m/s) (R2 m/s)...(Rn m/s Rninverse m/s)). If Rinverse is not given, R is defined as a property. The m/s is either M or S and determines whether the relation is multiple or single valued. For example, (DEFRELS (MEMBER/OF S HAS/MEMBER M) (COLOR S)) defines MEMBER/OF and HAS/MEMBER as inverse relations and COLOR as a property. MEMBER/OF is singular (i.e. an item is the member of only one thing) while HAS/MEMBER is multiple (i.e. an item can have many members). Defining a relation to be simple valued allows a savings of one CONS cell per link and prevents the user from inadvertently assigning it more than one value. If the user is not interested in these points, he may define all of his relations as M.

The main function for adding information to the network is ADDITEM. ADDITEM can be used to add new information about an existing item or to create new items. ADDITEM has the disadvantage that any items created by it must be "terms" (have LISP atoms which correspond to them). To create items without PNAMEs there is a similar function,

IBUILD. There is also a function ICONNECT which is useful for linking together groups of already defined items with a single relation.

The retrieval of information from the net can occur at several levels. The function IFIND which was described earlier can be used for complex retrievals from the net. The function ICONNECT? is useful for determining whether or not two sets of items are anywhere related by a particular relation. At the lowest levels, RELFOL returns the relations of an item while PROPFOL retrieves properties.

There are a large number of functions available for editing information in the network. To delete information about an item the function DELITEM is used. It provides ways of deleting particular links between items, removing all of those links of a particular relation that an item has, or deleting an item entirely. This last action which can also be done by function FREENODE puts the item cell back on the free item list. The functions CHANGEREL and CHNGPROP provide ways of changing respectively, the relation between two items or the properties of an item. To change the PNAME of an item there is a function RETITLE. In addition, a way is provided for calling the BBN LISP editor on either the property list of an item (IEDITP) or the list of relations of an item (IEDITR). IEDITR must be used with caution as all of the information on the relation list of an

item is ordered and is duplicated on the relation lists of the items pointed to by it and IEDITR makes no attempts to change the inverse links.

To print out information from the network there are three basic functions. The simplest, PRINTITEM, prints the name associated with an item. If the item has a PNAME or an EGO, that is what is printed. Otherwise, the item (the number) itself is printed. The function DESCRIBE (abbreviated D) prints out all of the arcs (properties and relations) leaving an item. To make a copy of the entire network, there is a function PNET. PNET prints a directory of the relations and properties, the terms and the egos of the network as well as DESCRIBEing each item. To save a copy of the network for later use, as opposed to later reading, the user should use the LISP function MAKEFILE on the file NET. More detail about the implementation of the network is available in the listings in Appendix 5.

Chapter 7

Conclusion

Introduction

The prior chapters have provided a rather comprehensive technical description of the top level details of Sophie. Because of the complexity of this kind of system these chapters may have raised more questions than they answered. The purpose of this chapter is to anticipate and answer some of these questions and to indicate some new dimensions along which Sophie might be profitably expanded.

It is important to realize that we do not view Sophie as a stand-alone CAI system but instead we view it as a powerful adjunct to efficient CAI delivery systems such as PLATO, AIS, etc. To use Sophie to spew out textual material typically contained in frame oriented systems would be a maximal misuse of Sophie's power. It seems best to let a PLATO type system perform those functions for which it has been carefully tuned and then to invoke Sophie (possibly via the ARPA network) to handle deeper "problem-solving sessions" with the student. In particular, by using Sophie an author can easily present tasks or questions to a student without being constrained to use only those tasks which possess simple extensionally defined answer sets. Likewise, with Sophie the author need not worry about selecting or

providing the detailed feedback for each student's individualized solution to a given task. Such uses make optimal use of Sophie's intelligence and as Sophie gets smarter (incorporating more knowledge and inference procedures) the more profound and individualized the analyses it performs will be.

Generality

Some of the most frequently asked questions about Sophie concern its generality and the closely associated issue of what an author must do to add new circuits to Sophie's repository of knowledge. When we started designing this system we purposely chose to model one and only one circuit feeling it would be unwise to build in the capability of handling multiple circuits until we had solved the major stumbling blocks of successfully handling one circuit. However the decision to model the IP-28 power supply was partially based on our concern for generality. Although many of our techniques for handling the power supply are currently limited to DC techniques, this particular power supply critically depends on encoding the functions of its feedback paths. Having found viable techniques for coping with feedback in itself guarantees us one important dimension of generality. Not only do feedback circuits permeate every moderately complicated electronic instrument but it is precisely feedback which raises havoc

with simplistic troubleshooting techniques (or logics) and which causes hopeless problems with classical inferencing techniques.

Adding New DC Circuits

Since much of Sophie's inferential capabilities stem from "specialists" intelligently manipulating and interpreting a general purpose simulation system, adding new DC circuits basically involves communicating a complete circuit description to the simulator. This could be done by storing the description of the new circuit in the semantic net. Since the various specialists are driven by the relevant portions of the net, the specialists themselves should require only minor changes. In addition, the semantics of the natural language processor would have to be enlarged to cope with whatever new concepts are present in the circuit and to be able to determine the proper denotation for terms used in asking questions about the circuit. (Handling several circuits simultaneously would require more profound changes but would surely be doable within the current framework of Sophie).

The only major stumbling block for adding a new DC circuit would concern the hypothesis generation specialists which make extensive use of the fast circuit-dependent simulator. Currently this simulator would require extensive

recoding for modelling most new circuits. Of course this problem can be circumvented by deactivating certain specialists but that action would eliminate most of the deductive capabilities of the "HELP" command. However, we think it might be possible to automate the construction of a circuit dependent simulator by developing some automatic programming techniques coupled with explicit knowledge derivable from using the general purpose simulator system to model the circuit under investigation. We see such research as providing a challenging new direction to the concepts of author aids and generative CAI.

Handling More Complicated Circuits

Our approach to handling more complicated circuits will use hierarchical functional models falling back on the individual component level of simulation for the "terminal" nodes. We expect no "terminal" to be any more complicated than our current circuit and therefore we should be able to use the general purpose simulator for modelling the lowest level blocks. A great deal more research is required before we fully understand the best way to handle "faultable" hierarchical models. In particular we hope to develop more qualitative simulators for the functional blocks which manifest some of the advantages of our automata-driven meteorological simulator (Bro73).

Handling AC Circuits

Although we could use our general purpose simulator for modelling AC aspects of a circuit, we again favor the development of a more qualitative approach which would merge the DC simulation techniques (for determining operating points, etc.) with AC specialists. These specialists would not only provide a qualitative simulation of the AC behavior but in conjunction with information derived from the DC simulation they could be used for the hypothesis generation or theory formation tasks.

New Avenues of Research

Although Sophie's present capabilities unquestionably demonstrate that AI techniques can have an immediate pay-off in CAI environments, we do not pretend that Sophie fully demonstrates such capabilities. In fact, we have just scratched the surface of what can be done with AI in an educational or training environment. What follows are some directions to be pursued using the basic framework set forth by Sophie.

1) Using deductive type specialists to create a commentary on, or a grading of, the sequence of measurements taken by the student.
   a) Creating a logical model of the information that an "ideal" student would have deduced from this sequence of measurements. This model could then be utilized by tutorial specialists to guide the student on remedial action and to explain to him why certain of his measurements were redundant, etc.

2) Expanding the question answering capabilities so that causal explanations can be given to "why" questions which follow "what happens if" type questions.

3) Building specialists that can deduce the appropriate boundary conditions to use in setting up the simulation runs for "if-then" questions and which can reveal why these conditions are interesting. (Current procedures perform simple inferencing of a SUPERC nature on the semantic net).

4) Building specialists which can perform qualitative reasoning about a new circuit and in particular, can "understand" what each component is doing in a given circuit, i.e. a set of specialists that can build a qualitative description of a circuit given only the semantics of the circuit.

5) Expanding Sophie's natural language capabilities to handle contextual problems in a unified way.

6) Incorporating some of the Scholar-type networks and inferencing procedures for answering generic types of questions such as "What is a transistor", etc.

7) Exploring how Sophie might benefit from certain distributive computation concepts wherein Sophie's simulator might run on a number-crunching computer with all the "smarts" sitting on top of the simulator residing in a LISP machine.

8) Exploring how to effectively interface Sophie with a PLATO type system.

Different Domains of Knowledge for Sophie

Although Sophie has been designed around electronic knowledge the underlying philosophy of Sophie (and its inferencing schema) can certainly be applied to other closed "worlds". For example, any domain for which simulation models exists would be obvious candidates. Several domains of knowledge which look particularly promising are:

1) programming wherein one can "simulate" or execute a program to verify some property about it and to understand some of the ramifications of a given malfunction or "bug"

2) mechanical systems

3) complex hydraulic systems

4) medical knowledge especially that pertaining to physiological or pharmacological processes

Not only do all these domains involve the teaching of troubleshooting or debugging techniques but they also lend themselves particularly well to powerful simulations. In fact many of the convergence problems plaguing simulation techniques in electronics are minimal in these domains. But of course there are other problems which are more difficult to solve in these domains. For example, in the programming area the inference techniques needed to derive "test data" (i.e. boundary conditions) are considerably more subtle than those in most electronic circuits.

Some AI Issues

Throughout this report we have tended to de-emphasize the AI aspects of this research except to note that AI techniques, judiciously used, can have an immediate impact on CAI technology. In these closing paragraphs we will sketch some of the novel and possibly significant AI concepts developed in the course of building Sophie.

The key to Sophie's deductive or inferential

capabilities lies in distributing intelligence across numerous specialists each of which incorporates domain dependent knowledge for carrying out its little task. Distributing the intelligence allows Sophie to easily use two drastically different means of encoding knowledge -- the "analogue" model and the deductive specialist -- each of which provides unique properties for expediting certain kinds of inference. The analogue model is especially well suited to account for the complex side effects and feedback type interactions underlying the consequences of certain actions. The deductive specialists are well equipped to handle "linear" reasoning where the consequences of an action do not cycle back and seriously alter the very state on which the deduction was initiated. In those situations where "feedback" leads to competing theories concerning what should override what, the analogue model can be used to simultaneously resolve all such conflicts. However, it often renders opaque those aspects which are well captured by the linear reasoning.*

An interesting AI issue lies in the interfacing of these different representations. Just as Gelenter's system used an analogue model of a theorem (in the form of a diagram) we use a simulation of a "theorem" (representing a

---

*A somewhat similar situation occurs in the trade-offs between time and frequency domains such as arise in Fourier analysis.

circuit) not only for guiding some of the procedural specialists but also for creating examples which provide kernel data for other specialists. In addition, Sophie incorporates heuristic "how to do it" type knowledge which is used to manipulate the model and then examine the results of this manipulation. It is precisely this kind of non-axiomatic manipulation which looks most promising in our view and has been one of the driving forces behind our concept of Sophie. In summary, we believe that a profound synergism can be obtained by correctly combining deductive models with "analogue" models.

Appendix 1

Instrument and Circuit Description of the IP-28 Power Supply

The IP-28 is a regulated power supply which has both current limiting and voltage control capabilities. (Schematic is included at the end of this appendix.) It is designed to deliver 1 amp at 3Ø volts when the current range switch (CR) is set to high and the voltage range switch is set to high. If CR is set to low then it delivers a maximum of Ø.1 amps and if VR is set to low it delivers a maximum of 1Ø volts. The current control (CC) and voltage control (VC) potentiometers regulate the cut off limits within the specified ranges. When VC is set to 1, the cut off limit is at the extreme high (e.g. 3ØV if VR=high) and if VC is set to Ø the output voltage should be limited to about 1 volt.

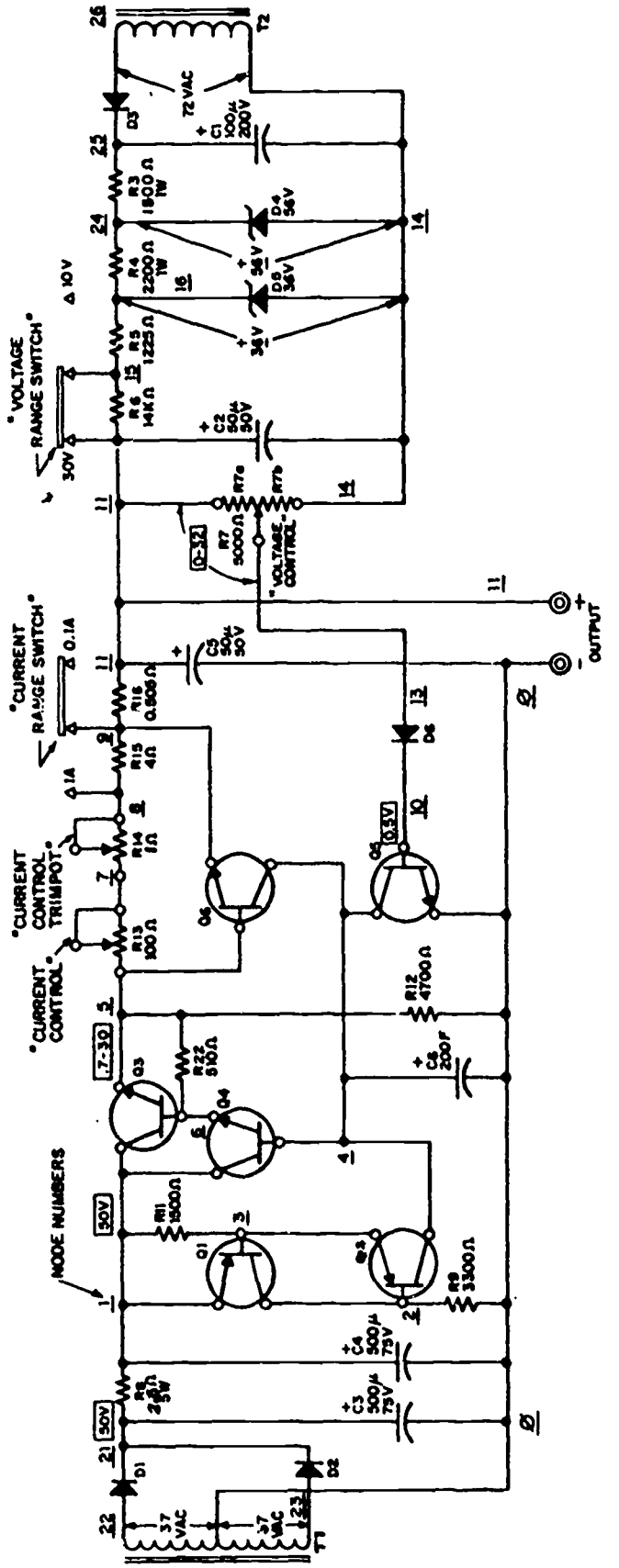
The internal DC supply voltage level is maintained by means of a full wave rectifier (diodes D1 and D2) connected to the center-tapped winding of the power transformer. A Pi configuration filter (resistor R8 and capacitors C3 and C4) reduces the ripple in the DC supply voltage to manageable proportions (about 4 volts AC remains). Transistors Q1, Q2 and resistors R9, R11 form a constant current source. This particular configuration gives good stability over voltage level variations twice as large as may be expected at the output of the filter, as well as assuring temperature

stability. The output of the constant current source drives the Darlington pair pass transistor consisting of transistors Q3 and Q4 and resistor R22. Q3, a power transistor, is capable of handling the desired currents (about 1 Amp), but is of relatively low beta. Q4's beta is very high (nominally 15Ø - 3ØØ); their combination assures both good capacity and quick response. R22 insures that the Darlington will cut off completely in situations where the collector-base leakage current of Q3 becomes significant. R12 is a steady state load which assures that Q3 does not cut off completely, and that minimum output voltage can be reached quickly.

Switching transistors Q5 and Q6 control the amount of current which actually flows into the base of Q4 from the output of the constant current source. Current limiting is achieved via resistor R15 and rheostats R14 (Current Control) and R13 (Trimpot). The voltage across this series resistance determines how much current Q6 will draw away from the base of the Darlington pair.

Voltage regulation is achieved via comparison of the output of the Darlington with the output of the reference voltage, the source of which is the half-wave rectifier on the other winding of the power transformer (diode D3). The output of the half wave rectifier is filtered by capacitor C1. Zener diode D4 insures that voltage will drop across R3

to a steady 56 volts. D5 and R4 act similarly to drop this voltage in turn to 36 volts. The use of two zener diodes insures that the output (final) zener will remain in a temperature stable region. The very clean reference voltage thus produced is applied across rheostat R7 (Voltage Control), one terminal of which is connected to the output of the power supply. The difference between the voltage drop in the reference branch of the circuit to the wiper of R7 and the voltage across the output load appears across the base-emitter junction of Q5, thereby controlling the amount of current that transistor will draw away from the base of the Darlington pair. C5 is used to reduce the output inpedance of the power supply, and C6 insures that the device will not oscillate due to switching transients.



Schematic of the IP-28 Power Supply

Appendix 2

BNF Description of the Grammar

This appendix gives a BNF-like description of the language accepted by Sophie. The grammar is implemented as LISP functions and some examples are listed in Appendix 3. The parsing process is sketched out and a list of compound words and abbreviations are given.

In the description, alternatives on the right hand side are separated by ! or are listed on separate lines. Brackets [] enclose optional elements. An asterisk * is used to mark notes about a particular rule. Non-terminals are designated by names enclosed in angle brackets <>.

The Grammar

```
<statement> := <request> ! <set> ! <modify>

<request> := <sim/request> [if <part/fault/spec>]
             if <part/fault/spec> [then] <sim/request>

<set> := set* <switch/spec> [to] <switch/value>
         set <pot/spec> [to] <pot/value>
         set <load/spec> [to] <load/value>
         put <load/value> [of] <load/spec>
         increase <pot/spec> [to] [<pot/value>]
         decrease <pot/spec> [to] [<pot/value>]
         *turn, put, let, switch, suppose and change also work.

<modify> := suppose* <part/fault/spec>
            replace** <part/spec>
            <fault/spec> <faultable/thing>
            <fault/up>
            clear <instrument/spec>
            fix <instrument/spec>
            remove fault
            reset <instrument/spec>
            restart
            *let and change also
            **switch and fix also.
```

<control/setting> := <switch/spec> ! <pot/spec> ! <load/spec>
[setting of] <control/setting>

<control/settings> := settings ! controls

<correct/mod> := correct ! working ! good ! normal
unfaulted ! proper ! new ! ok ! okay

<diode/spec> := <diode> ! <zener/diode>
<section> diode ! <section> zener/diode

<exial/var> := something ! anything

<fault/spec>:= open ! short ! burn/up ! blown ! leaky

<fault/up> := insert* [<fault/mod>] fault
insert fault <num/spec>
*use, introduce, enter and give also

<fault/mod> := hard ! easy ! benhaim

<faultable/thing> := <part/spec> ! <junction> ! <terminal>

<faulted/mod> := broken ! fault ! bad ! incorrect ! wrong ! problem
defective

<instrument/spec> := unit ! circuit ! instrument ! supply

<junction> := <junction/type> [of] <transistor/spec>
<transistor/term/type> and <transistor/term/type> [of]
<transistor/spec>
<transistor/term/type> to <transistor/term/type> [of]
<transistor/spec>

<junction/type> := eb ! be ! ec ! ce ! cb ! bc

<k/num/spec> := "number"k (i.e. 10k)

<load/spec> := load ! output resistor

```
<load/value> := <num/spec> [unit]
                <k/num/spec> [unit]

<meas/quant> := voltage ! current ! resistance* ! power
                *means measured resistance

<measurement/modifier> := <right/meas/mod> ! <wrong/meas/mod>

<measurement/pronoun> := it ! that [measurement]
                         that result ! that value

<measurement> := output <meas/quant> [of <transformer>]
                 <transformer> <meas/quant>
                 <meas/quant> between* <node> and* <node>
                 <meas/quant> of** <part/spec>
                 <meas/quant> between output terminals
                 <meas/quant> of <junction>
                 <meas/quant> of <node>
                 <meas/quant> from <junction>
                 <junction/type> <meas/quant> of <transistor/spec>
                 <transistor/term/type> <meas/quant> of
                   <transistor/spec>
                 *from-to also works
                 **at, thru, in, into, across and through also work

<model/modifier> := <correct/mod> ! <faulted/mod>

<model/spec> := in* <model/modifier> <instrument/spec>
                supposed [to] be
                *for and with also

<node> := <terminal> ! ground
          junction of <part/spec> and <part/spec>
          node between <part/spec> and <part/spec>
          [point] between <part/spec> and <part/spec>
          <node/name> ! [node] <node/number>

<node/number> := integer -1<n<27

<num/spec> := "any positive number" [k]

<part/fault/spec> := <faultable/thing> [is] <fault/spec>
                     <faultable/thing> [is] <model/modifier>
```

11.

<part/spec> [is] <part/value>
<exial/var> <faulted/mod> [<faultable/thing>]
<part/prop> <part/spec> <model/modifier>
<part/prop> <part/spec> <part/value>
<faultable/thing> has <faulted/mod> <part/prop>
<faultable/thing> has <part/prop> <part/value>

<part/spec> := <part/name> ! <load/spec> ! <section> <part/type>

<part/prop> := value ! beta ! break-down/voltage ! spec
leakage/resistance ! power/rating ! resistance*
*means specification value of resistance

<part/value> := <load/value> ! high ! low

<pot/spec> := cc ! vc ! cct

<pot/value> := <num/spec> ! on ! off ! low ! med! high ! max ! min

<right/meas/mod> := right ! correct ! reasonable ! ok ! okay
normal

<sim/request> := what is <measurement> [<model/spec>]
<model/spec> what is <measurement>
what is <simple/fact>[<model/spec>]
what is <measurement/pronoun> <model/spec>
what is <control/setting>
what are <control/settings>
what are specs <part/spec>
what should <measurement/pronoun> be
what should <measurement> be
what should <simple/fact> be
is <part/fault/spec>
is <measurement> <measurement/modifier>
is <measurement/pronoun> <measurement/modifier>
what happen [<measurement>]
is it possible [that] <part/fault/spec>
could the problem be [that] <part/fault/spec>
could it be [that] <part/fault/spec>
i think [that] <part/fault/spec>
what is wrong
what could be wrong
help
review [everything]
save

```
<simple/fact> := <part/prop> <part/spec>

<switch/spec> := vr ! cr ! standby

<switch/value> := low ! high ! on ! off ! 10v ! 10volt ! 30v ! 30volt
                  1amp ! .1amp ! 1a ! .1a ! min ! max
                  30 ! 10 ! 1 ! .1

<terminal> := output [terminal] ! <transistor/term> ! center/tap
              positive terminal <part/spec>
              negative terminal <part/spec>
              arode <diode/spec> ! cathode <diode/spec>
              wiper <pot/spec>

<transformer> := t1 ! t2

<transistor/spec> := <transistor> ! <section> transistor

<transistor/term> := <transistor/term/type> <transistor/spec>

<transistor/term/type> := base ! collector ! emitter

<wrong/meas/mod> := wrong ! incorrect
```

<transistor>, <capacitor>, <diode>, <resistor> and <zener/diode> all check the semantic network and parse correct part names, e.g. r9, q6.

<section> uses the semantic network to determine if a word is a section of the unit, e.g. current/limiter.

<part/name> uses the semantic network to see if a word is the name of a part e.g. r6, c4, t2.

<node/name> checks semantic network for node names.

The Parsing Process

The parsing is preceded by a prescan which expands abbreviations, does spelling correction on commonly misspelled words and recognizes compound words and inserts the correct slashes; for example, the user can type current control instead of CC or zener diode instead of zener/diode.

Any statement which does not parse normally is treated as if it were a "what is" type of question and another attempt is made by the parser, e.g. "beta of Q5" is treated as "what is the beta of Q5". If the statement still does not parse, an extended spelling correction is done.

Following is a list of the compound words and abbreviations recognized by the parser.

compound	abbreviations
Base Emitter -- BE	COLL
Emitter Base -- EB	EMIT
also CE, EC, BC, CB	I
Break Down Voltage	IB
Burn Up	IC
Center Tap	IE
Constant Current Source	IBE
Current Control	IEB
Current Control Trimpot	IEC
Current Limiting	ICE
Current Range Switch	ICB
Current Reference Source	IBC
Leakage Resistance	O or OP
Output Stage	OI
Pass Transistor	OV
Power Rating	R
Power Transformer	V
Reference Transformer	VBE
Voltage Control	VEB
Voltage Limiting	VEC
Voltage Range Switch	VCE
Voltage Source	VBC
Zener Diode	VCB
	WHAT's
	BDV
	LEAK

Appendix 3

Program Listing of the Grammar

This appendix provides some examples of the grammar functions. Included are most of the rules necessary to parse (recognize and semantically interpret) any occurrence of a terminal, i.e. "base of Q2" or "wiper of CC".

```
(<TRANSISTOR/TERM>
  [LAMBDA (STR)                    (* SPECIFICATION of A
                                   particular TRANSISTOR
                                   terminal.)
    (PROG (TS1 R1)
          (RETURN (COND
                    ((SETQ TS1 (<TRANSISTOR/TERM/TYPE>
                                STR))
                      (SETQ R1 RESULT)
                      (AND (SETQ STR (<TRANSISTOR/SPEC>
                                      (CDR TS1)))
                           (SETQ RESULT (LIST R1 RESULT))
                           STR])
```

```
(<TRANSISTOR/TERM/TYPE>
  [LAMBDA (STR N)                  (* Types of TRANSISTOR
                                   terminals.)
    (CHECKLST STR (QUOTE (BASE COLLECTOR EMITTER))
              N])
```

```
(<TRANSISTOR/SPEC>
  [LAMBDA (STR)                    (* Ways of specifying A
                                   particular TRANSISTOR.)
    (OR (<TRANSISTOR> STR)
        (<SECTION-PART> STR (QUOTE (TRANSISTOR])
```

```
(<SECTION-PART>
  [LAMBDA (STR PRLST)
          (* Looks for A REFERENCE to A PART via
          THE SECTION it is in.
          PRLST is THE PART types acceptable for
          THE particular occurrence.)
    (PROG (R1)
          (RETURN (AND (SETQ STR (<SECTION> STR))
                       (SETQ R1 RESULT)
                       (SETQ STR (CHECKLST (CDR STR)
                                           PRLST))
                       (SETQ RESULT (LIST (QUOTE FINDPART)
                                          R1 RESULT))
                       STR])


(<PART/SPEC>
  [LAMBDA (STR)
    (OR (<PART/NAME> STR)
        (<LOAD/SPEC> STR)
        (<SECTION-PART> STR
          (QUOTE (CAPACITOR DIODE RESISTOR TRANSISTOR
                            ZENER/DIODE])


(<TERMINAL>
  [LAMBDA (STR)                (* ALL REFERENCE to A terminal
                               of A PART.)
    (PROG (TS1 R1)
          (RETURN
            (COND
              ([SETQ TS1 (GOBBLE (CHECKWRD STR
                                           (QUOTE OUTPUT))
                                 (QUOTE (TERMINAL]
                (SETQ RESULT (QUOTE P/OP))
                TS1)
              ((<TRANSISTOR/TERM> STR))
              ((SETQ TS1 (CHECKWRD STR (QUOTE CENTER/TAP)))
                (SETQ RESULT (QUOTE M/T1<S>))
                TS1)
              ([SETQ TS1 (CHECKLST STR (QUOTE (POSITIVE
                                               NEGATIVE]
                (SETQ R1 RESULT)
                (AND
                  [SETQ TS1
                    (<PART/SPEC>
                      (CDR (GOBBLE TS1 (QUOTE (OF]
                (SETQ RESULT (LIST R1 RESULT))
                TS1]))
```

```
[[SETQ TS1 (CHECKLST STR (QUOTE (ANODE
                                 CATHODE]
  (SETQ R1 RESULT)
  (AND
     SETQ TS1
      (<DIODE/SPEC>
        (CDR (GOBBLE TS1 (QUOTE (OF]
    (SETQ RESULT (LIST R1 RESULT]
([SETQ TS1 (CHECKLST STR (QUOTE (WIPER]
  (AND (SETQ TS1 (<POT/SPEC> (CDR TS1)))
       (SETQ RESULT (LIST (QUOTE WIPER)
                          RESULT))
       TS1])
```



```
(CHECKLST
  [LAMBDA (STR LST N)
          (* Looks for one of A list of wrds
          (LST) within N words in THE STR.
          If found RESULT is set to THE word
          found.)
    (PROG NIL
          (OR N (SETQ N FUZZINESS))
      LP  (COND
            ((OR (NULL STR)
                 (ZEROP N))
              (RETURN))
            ((FMEMB (CAR STR)
                    LST)
              (SETQ RESULT (CAR STR))
              (RETURN STR)))
          (SETQ STR (CDR STR))
          (SETQ N (SUB1 N))
          (GO LP])
```

Appendix 4

Examples of Semantic Forms

These are some examples of sentences handled by the Natural Language Processor. Under each statement the semantic interpretation is given. The semantic interpretation is a function call which when executed performs the processing required by the statement.

Requesting measurements: (parse times including semantic interpretation are placed in parentheses)

What is the voltage across the base emitter junction of the current limiting transistor? (140 ms)
(MEASURE VOLTAGE (FINDPART CURRENT/LIMITER TRANSISTOR) BE)

What is the VBE of Q6? (120 ms)
(MEASURE VOLTAGE Q6 BE)

What is current thru the base of Q5? (130 ms)
(MEASURE CURRENT (BASE Q5))

What is the IB of Q5? (100 ms)
(MEASURE CURRENT Q5 BASE)

What is the output voltage? (80 ms)
(MEASURE VOLTAGE LOAD)

What is the voltage between node 1 and the positive terminal of C6? (280 ms)
(MEASURE VOLTAGE N1 (POSITIVE C6))

What is the dynamic resistance of R11? (120 ms)
(MEASURE RESISTANCE R11)

What is the power rating of R8? (100 ms)
(CHECKSPEC POWER/RATING R8)

What is the beta of the voltage limiting transistor? (110 ms)
(CHEKSPEC BETA (FINDPART VOLTAGE/LIMITER TRANSISTOR))

What are the specs of Q3? (90 ms)
(CHECKSPEC SPEC Q3)

In a working circuit what is the output voltage of the power reference transformer? (90 ms)
(MODELEVALQ (MEASURE VOLTAGE T2) GOOD)

Modifying the instrument:

Change the output load to 1Ø megohms
(STQ LOAD 1.ØE7)

Suppose the beta of Q5 is 2ØØ
(DOFAULT Q5 2ØØ BETA)

Suppose the breakdown voltage of D5 is 3Ø volts
(DOFAULT D5 3Ø BREAK-DOWN/VOLTAGE)

Let C2 be leaky
(DOFAULT C2 LEAKY)

Turn up the voltage control
(INCREASE VC)

Set the voltage range switch to 3Ø volts
(STQ VR HIGH)

Set the current control to maximum
(STQ CC 1.Ø)

Suppose the BE junction of Q6 is shorted
(DOFAULT (BE Q6) SHORT)

Insert a hard fault
(INSERTFAULT HARD)

Noun phrase utterances: (noun phrases get interpreted as questions)

Voltage between the base of Q5 and the wiper of the voltage control
(MEASURE VOLTAGE (BASE Q5) (WIPER VC))

Output voltage
(MEASURE VOLTAGE LOAD)

VBE of Q6
(MEASURE VOLTAGE Q6 BE)

I thru C6
(MEASURE CURRENT C6)

Miscellaneous Questions:

Is the current limiting transistor bad
(SEEFAULT (FINDPART CURRENT/LIMITING TRANSISTOR) BAD)

Is it possible that the breakdown voltage of D5 is too low
(TESTFAULT D5 LOW BREAK-DOWN/VOLTAGE)

Could the problem be that C2 is leaky
(TESTFAULT C2 LEAKY)

Is anything wrong?
(SEEFAULT EXIALVAR BAD)

What is wrong?
(LISTFAULTS)

If the EB of Q5 opens what is the voltage at node 4
(IFTHEN ((BE Q5) OPEN) (MEASURE VOLTAGE N4))

What happens when the current range switch opens
(IFTHEN (CR OPEN) (MEASURE VOLTAGE LOAD))

What could be wrong?
(HYPHELP)

Miscellaneous commands:

Replace resistor R6
(REPLACE R6)

Review
(REVIEW)

Remove all faults
(CLEARUNIT)

Reset the instrument
(RESETUNIT)

Restart
(RESTART)

Save
(SAVESYS)

Appendix 5

Program Listing of Semantic Network Functions

This appendix is a listing of the functions necessary to build and manipulate the semantic network used by Sophie. It is included in this report in the hope that others may find this package useful for building their networks. These functions can be obtained on Dectape or via the ARPA network from the authors.

```
(DEFINEQ

(**INTRO
  [LAMBDA NIL

          (* This is a collection of LISP routines which
          implement a semantic network patterned after
          Shapiro,
          "A NET STRUCTURE FOR SEMANTIC INFORMATION STORAGE"
          in IJCAI 71)


          (* The net is a collection of items
          (which are physically small numbers used as pointers
          into ITEMARRAY) linked together by RELATIONS and
          which may have properties. RELATIONS are pointers to
          other items in the NET and are stored in the CAR of
          the array cell. Properties are pointers to objects
          outside the NET and are stored in the CDR of the
          array cell. Terms are lisp atoms which provide entry
          points into the NET via the property SREF on their
          property list. These are kept on a global variable
          TERMS. EGO-DICT is a list of print name associated
          with items in the NET which are used to improve the
          readability of the NET.)


          (* A network can be saved on dsk by initially doing
          a LOAD on NET.INIT, evaluating
          (NEWNET n) and then doing a MAKEFILE on NET whenever
          a new copy of the network is desired.
          NETVARS contains the relevant information about
          saving the network.)


          (* some accessing of information in the network is
          performed by the function IFIND which is
          block-compiled and is on a separate file, IFIND.COM
          (which should be loaded before SEM.))
```

```
(*REL
  [LAMBDA (R)                                    (* Get the inverse of R)
    (OR (CDR (GETREL R))
        (HELP R "DOES NOT HAVE A RELATIONAL INVERSE."])

(ADDITEM
  [NLAMBDA ARG

          (* Adds information about an item or creates a new
          item. The form of a call is
          (ADDITEM t/i (r1 t/i11 t/i12 ...
          t/i1n) (r2 t/i21 t/i22 ... t/i2n) ...
          (rn t/in1 t/in2 ... t/inm)). If r1 is a relation,
          ADDITEM adds r1 LINKS between t/i and each t/iij.
          All t/i can be either terms or items.
          The t/i (ij), if not atomic can be a form which
          evaluates to a list of items.
          If r1 is a property, t/iij can BE anything and is
          stored as is.)


    (IBUILD1 [COND
               ((NUMBERP (CAR ARG))
                 (CAR ARG))
               ((SREF (CAR ARG)))
               (T (ADDTERM (CAR ARG)
                           (NEWITEM]
             (MAPCONC (CDR ARG)
                      (FUNCTION (LAMBDA (GRELSPEC)
                          (MAPCAR (CDR GRELSPEC)
                                  (FUNCTION (LAMBDA (Y)
                                      (LIST (CAR GRELSPEC)
                                            Y]))

(ADDRFL
  [LAMBDA (ITEMA R ITEMB)                        (* Add a 2-way link)
    (PUTLINK ITEMA R ITEMB)
    (PUTLINK ITEMB (*REL R)
             ITEMA])

(ADDTERM
  [LAMBDA (ATM ITEM)                             (* Adds a term and
                                                 connects it up to a
                                                 semantic referent)
    [OR (FMEMB ATM TERMS)
        (SETQ TERMS (MERGE TERMS (CONS ATM]
    (PUT ATM (QUOTE SREF)
         (PUTPROP ITEM (QUOTE PNAME)
                  ATM])
```

```
(ADDTOPROP
  [LAMBDA (NODE PROP PROPVAL)

          (* Adds properties to an item
          (properties are on the CDR, relations are on the
          CAR) if PROP is EGO, it updates EGO-DICT.)


    (PROG ((NODEPROPLIST (ELTD ITEMARRAY NODE))
           PVLS)
          [COND
            ((MULTP PROP)
              (SETQ PROPVAL (CONS PROPVAL)))
            ((EQ PROP (QUOTE EGO))
              (OR (MEMBER PROPVAL EGO-DICT)
                  (SETQ EGO-DICT (MERGE EGO-DICT (CONS PROPVAL]
          (COND
            ((NULL NODEPROPLIST)
              (SETD ITEMARRAY NODE (LIST PROP PROPVAL)))
            ((NULL (SETQ PVLS (GET NODEPROPLIST PROP)))
              (NCONC NODEPROPLIST (LIST PROP PROPVAL)))
            ((MULTP PROP)
              (NCONC PVLS PROPVAL))
            (T (HELP (QUOTE "ATTEMPT TO LINK SECOND SECOND VALUE")
                     PROP]))

(CHANGEREL
  [LAMBDA (FROM TO OLDREL NEWREL)

          (* CHANGEREL is used to edit the semantic network,
          i.e. to change the type of link between nodes from
          and to. OLDREL is the old link, while NEWREL is the
          new one.)


    (DELREL (SETQ FROM (SEMNODE FROM))
            OLDREL
            (SETQ TO (SEMNODE TO)))
    (ADDREL FROM NEWREL TO])

(CHNGPROP
  [LAMBDA (NODE PROP TO)
    (COND
      ([SETQ NODE (MEMB PROP (ELTD ITEMARRAY (SEMNODE NODE]
        (RPLACA (CDR NODE) TO))
      (T (HELP (QUOTE "NO PROPERTY")
               PROP])

(DEFREL
  [LAMBDA (R1 SM1 R2 SM2)

          (* DEFREL is used to define new primitive RELATIONS
          in the semantic network. It has been changed from
          kaplan's version to allow the use of one way LINKS
          or flags (properties))
```

```
    (COND
      ((GETREL R1 T)
        (ERROR R1 "ALREADY DEFINED AS A RELATION."))
      ((GETREL R2 T)
        (ERROR R2 "ALREADY DEFINED AS A RELATION."))
      ([OR [NOT (FMEMB SM1 (QUOTE (S M]
           (AND P2 (NOT (FMEMB SM2 (QUOTE (S M]
        (ERROR "RELATION SPECIFICATION HAS WRONG MULTIPLICITY SPEC.")))
    (PUT R1 (QUOTE REL)
         (CONS SM1 R2))
    (SETQ RELATIONS (CONS R1 RELATIONS))
    (COND
      (P2 (PUT R2 (QUOTE REL)
               (CONS SM2 R1))
          (SETQ RELATIONS (CONS R2 RELATIONS]))

(DEFRELS
  [NLAMBDA RELDEFS

          (* Used for defining relations ie
          (DEFRELS (r1 m/s r1inverse m/s)
          (r2 m/s r2inverse m/s) ...
          (rn m/s rninverse m/s)) if rinverse is not given, ri
          is a property.)


    (MAPC RELDEFS (FUNCTION (LAMBDA (RELDEF)
              (APPLY (FUNCTION DEFREL)
                     RELDEF]))

(DEGOS
  [LAMBDA (FILE SORTFLAG)

          (* DEGOS is used to print out a dictionary of the
          EGO LINKS for ease in looking through the semantic
          NET. It is called by PNET.)


    (PRIN1 "EGOS:" FILE)
    (TERPRI FILE)
    (MAPC (COND
            (SORTFLAG (SORT EGO-DICT T))
            (T EGO-DICT))
          (FUNCTION (LAMBDA (EGO)
              (SPACES 2 FILE)
              (PRIN1 EGO FILE)
              (TERPRI FILE]))
```

```
(DELITEM
  [NLAMBDA ARG

          (* Analogous to ADDITEM for deleting links.
          ARG can BE (ITEM), (ITEM PROP),
          (ITEM REL), (ITEM REL ITEM2)
          (ITEM (rl1 i1 I2) (rl2) (PROP)
          (rl4 i1 I2 i3 ...) ...). At present all of a
          property is deleted. To do otherwise, use IEDITP.)


     (PROG (ITEM REL ITEM2)
           (OR [NUMBERP (SETQ ITEM (SEMNODE (CAR ARG]
               (ERROR ITEM "NOT AN ITEM."))
           [COND
             ((NULL (CDR ARG))
               (FREENODE ITEM))
             [(ATOM (SETQ REL (CADR ARG)))
               (COND
                 ((ONEWAY? REL)
                   (DELPROP ITEM REL))
                 ((SETQ ITEM2 (CADDR ARG))
                   (OR (NUMBERP (SETQ ITEM2 (SEMNODE ITEM2)))
                       (ERROR ITEM2 "NOT AN ITEM."))
                   (DELREL ITEM REL ITEM2))
                 (T (MAPC (RELFOL ITEM REL)
                          (FUNCTION (LAMBDA (I2)
                                    (DELREL ITEM REL I2]
             (T (MAPC (CDR ARG)
                      (FUNCTION (LAMBDA (X)
                           (COND
                             [(CDR X)
                               (MAPC (CDR X)
                                     (FUNCTION (LAMBDA (Y)
                                         (APPLY* (FUNCTION DELITEM)
                                                 ITEM
                                                 (CAR X)
                                                 Y]
                             (T (APPLY* (FUNCTION DELITEM)
                                        ITEM
                                        (CAR X]
           (RETURN ITEM]))

(DELPROP
  [LAMBDA (NODE PROP)

          (* Deletes a property from an item.
          If the property is PNAME or EGO, it also removes its
          value from TERMS or EGO-DICT.
          If nothing is left on the item arter deletion, the
          item is put on the freelist, FREENODE.)
```

```
    (PROG (T1)
          [COND
            [(EQ PROP (QUOTE PNAME))
              (COND
                ((EQ (SREF (SETQ T1 (PROPFOL NODE PROP)))
                     NODE)
                  (REMPROP T1 (QUOTE SREF))
                  (SETQ TERMS (REMOVE T1 TERMS]
            ((EQ PROP (QUOTE EGO))
              (SETQ EGO-DICT (REMOVE (PROPFOL NODE PROP)
                                     EGO-DICT]
          (SETD ITEMARRAY NODE (MAPCON
                  (ELTD ITEMARRAY NODE)
                  [FUNCTION (LAMBDA (NP)
                      (COND
                        ((EQ (CAR NP)
                             PROP)
                          NIL)
                        (T (LIST (CAR NP)
                                 (CADR NP]
                  (FUNCTION CDDR)))
          (FREEIFNULL NODE])

(DELREL
  [LAMBDA (*FROM REL *TO)

          (* DELREL is used to delete a link between two nodes
          in the semantic network, *FROM and *TO.
          Since every link is two-way, DELREL1 is called to
          delete both directed LINKS.)


    (DELREL1 (SETQ *FROM (SEMNODE *FROM))
             REL
             (SETQ *TO (SEMNODE *TO)))
    (DELREL1 *TO (*REL REL)*FROM)*FROM])

(DELREL1
  [LAMBDA (FROM REL TO)

          (* DELREL1 is called by DELREL to delete a directed
          link between nodes from and to.
          From and to are integers, pointing to nodes in the
          semantic network. If nothing is left on node from
          after the deletion, it is added to the pool of
          available nodes, FREENODE.)
```

```
    (PROG ((FLINKS (LINKS FROM))
           X)
          (OR (SETQ X (ASSOC REL FLINKS))
              (RETURN NIL))
          [COND
            [[EQ (QUOTE S)
                 (CAR (GETP REL (QUOTE REL]
              (COND
                [(EQ TO (CDR X))
                  (SETA ITEMARRAY
                     FROM (MAPCONC FLINKS (FUNCTION (LAMBDA (Y)
                                                    (AND (NEQ Y X)
                                                         (LIST Y]
                 (T (PRIN1 FROM)
                    (PRIN1 (QUOTE " IS NOT LINKED TO "))
                    (PRIN1 TO)
                    (PRIN1 (QUOTE " VIA "))
                    (PRINT REL)
                    (RETURN T]
              (T [SETQ X (CONS (CAR X)
                               (MAPCONC (CDR X)
                                        (FUNCTION (LAMBDA (Y)
                                            (AND (NEQ Y TO)
                                                 (LIST Y]
                 (SETA ITEMARRAY
                    FROM (MAPCONC FLINKS (FUNCTION (LAMBDA (Y)
                                        (COND
                                          ((EQ (CAR X)
                                               (CAR Y))
                                            (AND (CDR X)
                                                 (LIST X)))
                                          (T (LIST Y]
          (FREEIFNULL FROM])

(DESCRIBE
  [NLAMBDA ITEMS                                    (* Prints the arcs
                                                    leaving item/terms)
    (MAPC (COND
            ((NULL ITEMS)
              LASTITEMS)
            ((NLISTP ITEMS)
              (LIST ITEMS))
            (T ITEMS))
          (FUNCTION DESCRIBE1])

(DESCRIBE1
  [LAMBDA (NODE)

          (* DESCRIBE1 describes all the arcs leaving and
          entering item. NODE is either a literal atom or a
          number. If a literal atom does not correspond to a
          NODE in the semantic network
          (i.e. it does not have an SREF property), DESCRIBE1
          returns NIL. If a NODE is in the pool of free nodes
          and has no LINKS, it is not printed.)
```

```
    (COND
      [(NULL (SETQ NODE (SEMNODE NODE]
      ((OR (LINKS NODE)
           (LISTP (ELTD ITEMARRAY NODE)))
        (PRIN1 NODE)
        (TERPRI)
        [COND
          ((PROPFOL NODE (QUOTE PNAME))
           (SPACES 3)
           (PRIN1 (QUOTE PNAME))
           (TAB 14 1)
           (PRINT (PROPFOL NODE (QUOTE PNAME]
        (PRINTPROPS NODE)
        (MAPC (LINKS NODE)
              (FUNCTION (LAMBDA (RELSPEC)
                  (SPACES 3)
                  (PRIN1 (CAR RELSPEC))
                  (TAB 14 1)
                  [COND
                    [(MULTP (CAR RELSPEC))
                      (MAPC (CDR RELSPEC)
                            (FUNCTION (LAMBDA (X)
                                (COND
                                  ((ILESSP 45 (POSITION))
                                    (TAB 14 1)))
                                (PRINTITEM X)
                                (SPACES 1]
                    (T (PRINTITEM (CDR RELSPEC]
                  (TERPRI]))

(DITEMS
  [LAMBDA NIL                                    (* Prints all of the
                                                 items in ITEMARRAY)
    (PROG ((N 0))
          (PRIN1 "ITEMS:")
          (TERPRI)
          [RPTQ NITEMS (DESCRIBE1 (SETQ N (ADD1 N]
          (TERPRI)
          (RETURN T])
```

```
(DRELS
  [LAMBDA (FILE)                                    (* Prints all of the
                                                    RELATIONS)
    (PRIN1 "RELATIONS:" FILE)
    (TERPRI)
    (MAPC RELATIONS (FUNCTION (LAMBDA (RELATION)
              (SPACES 2 FILE)
              (PRIN1 RELATION FILE)
              (TAB 12 1 FILE)
              (PRIN1 (CAR (SETQ RELATION (GETREL RELATION)))
                     FILE)
              (COND
                ((CDR RELATION)
                  (TAB 17 1 FILE)
                  (PRIN1 (CDR RELATION)
                         FILE)
                  (TAB 27 1 FILE)
                  (PRINT (CAR (GETREL (CDR RELATION)))
                         FILE))
                (T (TERPRI FILE])

(DTERMS
  [LAMBDA (SORTFLAG)

          (* Prints all of the TERMS and the item number of
          their semantic referent.)


    (PRIN1 "TERMS:")
    (TERPRI)
    [MAPC (COND
            (SORTFLAG (SORT (COPY TERMS)))
            (T TERMS))
          (FUNCTION (LAMBDA (TERM)
              (SPACES 2)
              (PRIN1 TERM)
              (TAB 16 1)
              (PRINT (SREF TERM]
    (TERPRI])

(EXPANDNET
  [LAMBDA (NEWSIZE)                                 (* Copy net into an
                                                    enlarged array)
    (PRIN1 "***ENLARGING NET TO " T)
    (PRIN1 NEWSIZE T)
    (PRINT1 " ITEMS." T)
    (PROG ((NEW (ARRAY NEWSIZE))
           I)
      LP  [COND
            ((EQ I NITEMS)
              (SETQ ITEMARRAY NEW)
              (RETURN (SETQ NETSIZE NEWSIZE]
          (SETA NEW I (ELT ITEMARRAY I))
          (SETD NEW I (ELID ITEMARRAY I))
          (SETQ I (ADD1 I))
          (GO LP])
```

```
(FREEIFNULL
  [LAMBDA (NODE)

          (* Checks and frees an item if it has neither
          properties nor RELATIONS. FREENODE is the freelist.)


    (OR (ELTD ITEMARRAY NODE)
        (ELT ITEMARRAY NODE)
        (PROGN (SFTD ITEMARRAY NODE FREENODE)
               (SETQ FREENODE NODE])

(FREENODE
  [LAMBDA (NODE)                                    (* Deletes a NODE by
                                                    deleting all of its
                                                    RELATIONS and
                                                    properties.)
    [MAPC (LINKS NODE)
          (FUNCTION (LAMBDA (X)
              (APPLY* (FUNCTION DELITEM)
                      NODE
                      (CAR X]
    (MAP (ELTD ITEMARRAY NODE)
         [FUNCTION (LAMBDA (X)
             (DELPROP NODE (CAR X]
         (FUNCTION CDDR])

(GETREL
  [LAMBDA (R OKFLAG)
    (COND
      ((NULL R)
        NIL)
      ((GFTP R (QUOTE REL)))
      (OKFLAG NIL)
      ((HELP R "IS NOT A DEFINED RELATION.")
        (GETREL R OKFLAG])

(IBUILD
  [NLAMBDA ARGS

          (* Builds a new item from a specification list, ie
          (IBUILD (PNAME name) (r1 t/i1)
          (r2 t/i2) (r1 t/i2) ...). Also see ADDITEM)


    (IBUILD1 (NEWITEM)
             ARGS])

(IBUILD1
  [LAMBDA (NEWITEM ARGS)

          (* Builds an ITEM from a specification.
          Returns a 'LIST' of the new ITEM)
```

```
(PROG (REL ITEMS)
      [MAPC
        ARGS
        (FUNCTION (LAMBDA (RELSPEC)
            (PROG (ANS)
                  (SETQ REL (CAR RELSPEC))
              L1  [COND
                    ((NOT (RELP REL))
                      (PRINT REL T)
                      (PRIN1 (QUOTE "IS MISSPELLED OR UNDEFINED")
                             T)
                      (TERPRI T)
                      (PRIN1
                        (QUOTE
                          "TYPE S TO CHANGE SPELLING, D TO DEFINE")
                        T)
                      (TERPRI)
                      (COND
                        ((EQ (SETQ ANS (READ T))
                             (QUOTE S))
                          (PRIN1 (QUOTE "CORRECT SPELLING IS  ")
                                 T)
                          (SETQ REL (READ T))
                          (GO L1))
                        ((EQ ANS (QUOTE D))
                          (PRIN1 (QUOTE "TYPE DEFINING FORM")
                                 T)
                          (TERPRI)
                          (EVAL (READ))
                          (GO L1))
                        (T (GO L1]
                    (SETQ ITEMS (CADR RELSPEC))
                    (COND
                      ((ONEWAY? REL)
                        (ADDTOPROP NEWITEM REL ITEMS))
                      [(ATOM ITEMS)
                        (COND
                          ((NUMBERP ITEMS)
                            (ADDREL NEWITEM REL ITEMS))
                          (T (ADDREL NEWITEM REL (SREF ITEMS T]
                      [[NUMBERP (CAR (SETQ ITEMS (EVAL ITEMS]
                        (MAPC ITEMS (FUNCTION (LAMBDA (ITEM)
                                    (ADDREL NEWITEM REL ITEM]
                      (T (HELP "ATTEMPT TO LINK NON-ITEM TO ITEM VIA"
                               REL]
      (RETURN (LIST NEWITEM])
```

```
(ICONNECT
  [NLAMBDA (ITEMA REL ITEMB)
    [SETQ ITEMA (COND
        ((ATOM ITEMA)
          (LIST (SREF ITEMA T)))
        (T (EVAL ITEMA]
    [SETQ ITEMB (COND
        ((ATOM ITEMB)
          (LIST (SREF ITEMB T)))
        (T (EVAL ITEMB]
    (MAPC ITEMA (FUNCTION (LAMBDA (A)
              (MAPC ITEMB (FUNCTION (LAMBDA (B)
                        (ADDREL A REL B])

(ICONNECTED?
  [NLAMBDA (ITEMA REL ITEMB)
    [SETQ ITEMA (COND
        ((ATOM ITEMA)
          (LIST (SREF ITEMA T)))
        (T (EVAL ITEMA]
    [SETQ ITEMB (COND
        ((ATOM ITEMB)
          (LIST (SREF ITEMB T)))
        (T (EVAL ITEMB]
    (SOME ITEMA (FUNCTION (LAMBDA (A)
              (SOME (RELFOL A REL)
                    (FUNCTION (LAMBDA (AA)
                        (MEMB AA ITEMB])

(IDIFF
  [NLAMBDA ARGS                                    (* Set difference of 2
                                                   itemlists.)
    (SDIFF (EVAL (CAR ARGS))
           (EVAL (CADR ARGS])

(IEDITP
  [NLAMBDA (ITEM)

          (* Allows editing of the property list
          (CDR) of a item/term. Cannot BE used to delete all
          of the properties of an ITEM.)


    [EDITE (ELTD ITEMARRAY (COND
                    ((NUMBERP ITEM)
                      ITEM)
                    ((SREF ITEM))
                    ((ERROR ITEM "NOT AN ITEM"]
    ITEM])

(IEDITR
  [NLAMBDA (ITEM)

          (* Allows editing of the relation list
          (CAR) of a item/term. Cannot BE used to delete all
          of the properties of an ITEM.)
```

```
    [EDITE (ELT ITEMARRAY (COND
                  ((NUMBERP ITEM)
                    ITEM)
                  ((SREF ITEM))
                  ((ERROR ITEM "NOT AN ITEM"]
    ITEM])

(LINKS
  [LAMBDA (ITEM)                                    (* Retrieve LINKS of
                                                    ITEM)
    (ELT ITEMARRAY ITEM])

(MULTP
  [LAMBDA (R)                                       (* Is R a multiple
                                                    relation?)
    (EQ (CAR (GETREL R))
        (QUOTE M])

(NEWITEM
  [LAMBDA NIL                                       (* Allocate a new item)
                                                    (* Enlarging the net if
                                                    necessary)
    (COND
      (FREENODE (PROG ((I FREENODE))
                      (SETQ FREENODE (ELTD ITEMARRAY I))
                      (SETD ITEMARRAY I NIL)
                      (RETURN I)))
      (T (AND (EQ NITEMS NETSIZE)
              (EXPANDNET (ITIMES 2 NETSIZE)))
         (SETQ NITEMS (ADD1 NITEMS])

(NEWNET
  [LAMBDA (N)                                       (* Construct a new net)
                                                    (* EXPANDNET will be
                                                    used to expand the net
                                                    if it becomes full.)
    (SETQ ITEMARRAY (ARRAY N))
    (SETQ NETSIZE N)
    (SETQ NITEMS 0)
    (SETQ SUBLIST NIL)
    (SETQ FREENODE NIL)
    (SETQ TERMS NIL)
    (SETQ EGO-DICT NIL])

(ONEWAY?
  [LAMBDA (REL)
    (NULL (CDR (GETREL REL])

(PNAME
  [LAMBDA (ITEM)

          (* PNAME returns THE print NAME
          (PNAME) of AN ITEM. ITEM must BE A NUMBER.)
```

```
    (AND ITEM (SETQ ITEM (PROPFOL ITEM (QUOTE PNAME)))
         (OR (LISTP ITEM)
             (CONS ITEM])
```



```
(PNET
  [NLAMBDA (FILE)

          (* Prints the entire net on file FILE
          (relations, terms, egos and items) in a humanoid
          readable (but not LISP readable) form.
          To save the network use MAKEFILE
          (NET) --- see **INTRO)


    (PRETTYDEF NIL FILE (QUOTE ((E (DRELS)
                                   (DTERMS T)
                                   (DEGOS)
                                   (DITEMS])

(PRINT1
  [LAMBDA (XP)
    (PRIN1 XP)
    (TERPRI])

(PRINTITEM
  [LAMBDA (ITEM)

          (* PRINTITEM prints items in the semantic network.
          If the ITEM has a print name
          (PNAME) or an EGO link, that is what is printed.
          Otherwise, it is the ITEM itself.
          EGO LINKS are printed in square brackets to
          distinguish them from pnames which are in
          parentheses.)


    (OR (AND (NUMBERP ITEM)
             (PROPFOL ITEM (QUOTE PNAME))
             (PRINTDEF (PNAME ITEM)
                       (POSITION)))
        (AND (NUMBERP ITEM)
             (PROPFOL ITEM (QUOTE EGO))
             (PRIN1 (QUOTE "["))
             [NULL (MAPC (PROPFOL ITEM (QUOTE EGO))
                         (FUNCTION (LAMBDA (X)
                             (PRIN1 X)
                             (COND
                               ((ILESSP 65 (POSITION))
                                 (TAB 20 1))
                               (T (SPACES 1]
             (PRIN1 ITEM)
             (PRIN1 (QUOTE "]"))
            T)
        (PRINTDEF ITEM (POSITION])
```

```
(PRINTITEMLIST
  [LAMBDA (ITEMS FILE)
    (PROG ((PREFILE (OUTPUT FILE)))
          (MAPRINT ITEMS NIL "<" ">" " " (FUNCTION PRINTITEM))
          (TERPRI)
          (RETURN (OUTPUT PREFILE])

(PRINTPROPS
  [LAMBDA (NODE)
    (PROG ((PROPLIST (FLTD ITEMARRAY NODE)))
      LP  (COND
            ((NULL PROPLIST)
              (RETURN NIL))
            ((EQ (CAR PROPLIST)
                 (QUOTE PNAME)))
            (T (SPACES 3)
               (PRIN1 (CAR PROPLIST))
               (TAB 14 1)
               [COND
                 [(MULTP (CAR PROPLIST))
                   (MAPC (CADR PROPLIST)
                         (FUNCTION (LAMBDA (X)
                             (COND
                               ((ILESSP 45 (POSITION))
                                 (TAB 14 1)))
                             (PRINTDEF X (POSITION))
                             (SPACES 1]
                 (T (PRINTDEF (CADR PROPLIST)
                              (POSITION]
               (TERPRI)))
          (SETQ PROPLIST (CDDR PROPLIST))
          (GO LP])

(PROPFOL
  [LAMBDA (NODE PROP)
    (GET (ELTD ITEMARRAY NODE,
         PROP])

(PUTLINK
  [LAMBDA (FROM R TO)                          (* Add a 1-way link)
    (PROG (ITEMLIST (FLINKS (LINKS FROM)))
          [COND
            ((MULTP R)
              (SETQ TO (LIST TO]
          (COND
            [(NULL FLINKS)                     (* From has no LINKS)
              (SETA ITEMARRAY FROM (LIST (CONS R TO]
            ((NULL (SETQ ITEMLIST (RELFOL FROM R)))
                                               (* From has no r-links)
              (MERGE FLINKS (LIST (CONS R TO))
                     T))
            ((MULTP R)
              (MERGE ITEMLIST TO))
            (T (HELP R "SINGULAR, CAN'T PUT MULTIPLE LINK")))
          (RETURN FROM])
```

```
(PUTPROP
  [LAMBDA (NODE PROP PROPVAL)
    (SETD ITEMARRAY NODE (NCONC (LIST PROP PROPVAL)
                                (ELTD ITEMARRAY NODE)))
    NODE])

(RELFOL
  [LAMBDA (ITEM REL)                          (* Gets a list of items
                                              linked to an ITEM by a
                                              relation)
    (COND
      ([SETQ ITEM (CDR (ASSOC REL (LINKS ITEM]
        (COND
          ((MULTP REL)
            ITEM)
          (T (LIST ITEM])

(RELP
  [LAMBDA (REL)
    (GETP REL (QUOTE REL])

(RETITLE
  [LAMBDA (OLDNAME NEWNAME)

          (* RETITLE is an editing function for the semantic
          network. It permits one to change the print name of
          a node.)


    (PUT NEWNAME (QUOTE SREF)
         (SREF OLDNAME))
    (CHNGPROP (SREF NEWNAME)
              (QUOTE PNAME)
              NEWNAME)
    (REMPROP OLDNAME (QUOTE SREF))
    (RPLACA (MEMB OLDNAME TERMS)
            NEWNAME)
    (SORT TERMS)
    NEWNAME])

(SEMNODE
  [LAMBDA (NODE)

          (* SEMNODE is used by many functions to change their
          input into a pointer into the semantic network.
          If NODE, which is bound in the calling function,
          does not appear in the semantic network
          (i.e. have an SREF), SEMNODE causes an error.)
```

```
    (COND
      ((NULL NODE)
        NIL)
      [(LITATOM NODE)
        (COND
          ((SREF NODE))
          (T (ERROR "WORD NOT IN SEMNET - " NODE]
      (T NODE]))

(SREF
  [LAMBDA (ATM BUILDFLAG)                          (* Gets or builds a
                                                   semantic referent for an
                                                   atom (term))
    (COND
      ((GETP ATM (QUOTE SREF)))
      (BUILDFLAG (ADDTERM ATM (NEWITEM]))

(TOPFN
  [NLAMBDA (FN)                                    (* Does top-level print)
    (PRINTITEMLIST [SETQ LASTITEMS (APPLY FN (COND
                                               (LISPXLISTFLG LISPXLINE)
                                               (T (CAR LISPXLINE]
                   T]))
)
  (LISPXPRINT (QUOTE SEMFNS)
              T)
  (RPAQQ SEMFNS
         (**INTRO *REL ADDITEM ADDREL ADDTERM ADDTOPROP CHANGEREL
                  CHNGPROP DEFREL DEFRELS DEGOS DELITEM DELPROP DELREL
                  DELREL1 DESCRIBE DESCRIBE1 DITEMS DRELS DTERMS
                  EXPANDNET FREEIFNULL FREENODE GETREL IBUILD IBUILD1
                  ICONNECT ICONNECTED? IDIFF IEDITP IEDITR LINKS MULTP
                  NEWITEM NEWNET ONEWAY? PNAME PNET PRINT1 PRINTITEM
                  PRINTITEMLIST PRINTPROPS PROPFOL PUTLINK PUTPROP
                  RELFOL RELP RETITLE SEMNODE SREF TOPFN))
  (LISPXPRINT (QUOTE SEMVARS)
              T)
  (RPAQQ SEMVARS ((VARS SEMMACROS (LISPXMACROS (APPEND SEMMACROS
                                                       LISPXMACROS)))
         (P * SEMEXPRS)))
  [RPAQQ SEMMACROS ((F (TOPFN IFIND))
         (B (TOPFN IBUILD))
         (- (TOPFN IDIFF]
  (RPAQ LISPXMACROS (APPEND SEMMACROS LISPXMACROS))
  [RPAQQ SEMEXPRS ((EQUIVALENCE (F IFIND)
                                (B IBUILD)
                                (D DESCRIBE)
                                ($ DEFREL)
                                (- IDIFF)
                                (C ICONNECT)
                                (C? ICONNECTED?]
```

```
(EQUIVALENCE (F IFIND)
             (B IBUILD)
             (D DESCRIBE)
             (* DEFREL)
             (- IDIFF)
             (C ICONNECT)
             (C? ICONNECTED?))
STOP
```

```
(FILECREATED "12-DEC-73 11:54:20" IFIND

     changes to:  IFIND,ORDERSECT,ORDERUNION,RELFOL,IFINDVARS)


(DEFINEQ

(IFIND
  [NLAMBDA ARGS

          (* Retrieves information from the network according
          to a specification. For example
          (IFIND (^UNCLE ^JIM) (^SPOUSE
          (IFIND (^SIBLING ^MARY)))) would retrieve those
          ITEMS who are both uncles of JIM and married to one
          of MARY'S siblings.)


    (PROG (*REL ITEMS)
          (NINTERSECTION
            (MAPCAR ARGS
                    (FUNCTION (LAMBDA (RELSPEC)
                        (SETQ *REL (*REL (CAR RELSPEC)))
                        (SETQ ITEMS (CADR RELSPEC))
                        (COND
                          [(ATOM ITEMS)
                            (COND
                              ((EQ *REL (QUOTE SREF))
                                (LIST (SREF ITEMS)))
                              (T (RELFOL (SREF ITEMS)*REL]
                          (T (NUNION (MAPCAR (EVAL ITEMS)
                                             (FUNCTION (LAMBDA (ITEM)
                                               (RELFOL ITEM *REL])

(NUNION
  [LAMBDA (LISTS)                                    (* Brute force n-way
                                                     ORDERUNION)
    (COND
      [(CDR LISTS)
        (ORDERUNION (CAR LISTS)
                    (NUNION (CDR LISTS]
      (T (CAR LISTS])

(NINTERSECTION
  [LAMBDA (LISTS)                                    (* Brute force n-way
                                                     ORDERSECT)
    (COND
      [(CDR LISTS)
        (ORDERSECT (CAR LISTS)
                   (NINTERSECTION (CDR LISTS]
      (T (CAR LISTS])
```

```
(ORDERSECT
  [LAMBDA (L1 L2)

          (* Does a two-way INTERSECTION on lists which are
          ordered via ILESSP. The relations in the network are
          maintained in sorted order.)


    (PROG (R)
      LP  [COND
            ((OR (NULL L1)
                 (NULL L2))
              (RETURN (DREVERSE R)))
            ((EQ (CAR L1)
                 (CAR L2))
              (SETQ R (CONS (CAR L1)
                            R))
              (SETQ L1 (CDR L1))
              (SETQ L2 (CDR L2)))
            ((ILESSP (CAR L1)
                     (CAR L2))
              (SETQ L1 (CDR L1)))
            (T (SETQ L2 (CDR L2]
          (GO LP])

(ORDERUNION
  [LAMBDA (L1 L2)                                  (* Does a two-way UNION
                                                   of ordered lists.)
    (PROG (R)
      LP  [COND
            ((NULL L1)
              (RETURN (NCONC (DREVERSE R)
                             L2)))
            ((NULL L2)
              (RETURN (NCONC (DREVERSE R)
                             L1)))
            ((EQ (CAR L1)
                 (CAR L2))
              (SETQ R (CONS (CAR L1)
                            R))
              (SETQ L1 (CDR L1))
              (SETQ L2 (CDR L2)))
            ((ILESSP (CAR L1)
                     (CAR L2))
              (SETQ R (CONS (CAR L1)
                            R))
              (SETQ L1 (CDR L1)))
            (T (SETQ R (CONS (CAR L2)
                             R))
               (SETQ L2 (CDR L2]
          (GO LP])
```

```
(RELFOL
  [LAMBDA (ITEM REL)

          (* Gets those items linked to an ITEM via a
          particular relation. This version is stripped of
          error checking to increase the speed of IFIND.
          There is another version on the file SEM.)


    (COND
      ([SETQ ITEM (CDR (FASSOC REL (LINKS ITEM]
        (COND
          ((MULTP REL)
            ITEM)
          (T (LIST ITEM])
)
  (LISPXPRINT (QUOTE IFINDFNS)
              T)
  (RPAQQ IFINDFNS (IFIND NUNION NINTERSECTION ORDERSECT ORDERUNION
                         RELFOL))
  (LISPXPRINT (QUOTE IFINDVARS)
              T)
  (RPAQQ IFINDVARS ((BLOCKS * IFINDBLOCKS)))
  [RPAQQ IFINDBLOCKS ((IFINDBLK IFIND NUNION ORDERUNION NINTERSECTION
                                ORDERSECT RELFOL (ENTRIES IFIND)
                                (GLOBALVARS ITEMARRAY)
                                (LINKFNS . T]
(DECLARE
  (BLOCK: IFINDBLK IFIND NUNION ORDERUNION NINTERSECTION ORDERSECT
          RELFOL (ENTRIES IFIND)
          (GLOBALVARS ITEMARRAY)
          (LINKFNS . T))
)STOP
```

References

(Bob72) Bobrow, D.G., Burchfield, J.O., Murphy, D.L. and Tomlinson, R.S. "TENEX, a Paged Time Sharing System for the PDP-10", Communications of the ACM, March 1972

(Bro73) Brown, J.S., Burton, R.R. and Zdybel, F. "A Model-Driven Question Answering System for Mixed-Initiative Computer Assisted Instruction", IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-3, May 1973

(Car73) Carbonell, J.R. and Collins, A.M., "Natural Semantics in Artificial Intelligence," Proceedings of Third IJCAI, SRI Publications Department, Menlo Park, Calif. August 1973, pp. 344-351

(Dos71) Dostert, B.H. and Thompson, F.B., "The Syntax of REL English," REL Report No. 1, California Institute of Technology, Pasadena, Calif. September 1971

(Fox65) Fox, L., An Introduction to Numeric Linear Algebra, Oxford University Press, New York, 1965, pp. 60-65, 99-102

(Nag71) Nagel, L.W. "Transient Analysis of Large Non-linear Networks", IEEE Journal of Solid State Circuits, Vol. SC-6, August 1971

(Nag73) Nagel, L.W. and Pederson, D.O. "SPICE: Simulation Program with Integrated Circuit Emphasis", Memorandum ERL-M382, Electronics Research Laboratory, College of Engineering, University of California at Berkeley, April 1973

(Sha71) Shapiro, S.C. "A Network Structure for Semantic Information Storage, Deducation and Retrieval," Proceedings of Second IJCAI, 1971, pp. 512-523

(Tei74) Teitelman, W., INTERLISP Reference Manual, XEROX PARC, Palo Alto, Calif., February 1974

(Win73) Winograd, T., Understanding Natural Language, Academic Press, New York, 1973

(Woo72a) Woods, W.A., "An Experimental Parsing System for Transition Network Grammars," BBN Report 2362, Bolt Beranek and Newman, Inc. Cambridge, Mass. May 1972

(Woo72b) Woods, W.A., Kaplan, R.M. and Nash-Webber, B., "The Lunar Sciences Natural Language Information System: Final Report," BBN Report 2378, Bolt Beranek and Newman, Inc. Cambridge, Mass. June 1972

(Woo74) Woods, W.A. "Motivation and Overview of BBN SPEECHLIS: An Experimental Prototype for Speech Understanding Research", to appear in Proceedings of IEEE Symposium on Speech Recognition, CMU, April 1974